REMARKS

Claims 1-24 are pending in the Application, and all claims have been rejected in the Office action mailed October 2, 2007. The claims 2, 17, 20, 21, 23, and 24 are amended to remove the word "further" from "further comprising" or "further comprises." Claims 19, 20, and 22 are amended to correct typographical errors. Claim 22 is amended to clarify that registration of a call-back function is associated with a server. This is disclosed, for example, by claims 1 and 16, and paragraphs [0042] - [0046] and [0048] of the specification. Accordingly, the Applicant believes that the scope of the claims is not broadened by these amendments. Claims 1, 16, and 22 are independent claims. Claims 2-15, 17-21, and 23-24 depend from independent claims 1, 16, and 22, respectively.

The Applicant respectfully requests reconsideration of pending claims 1-24, in light of the following remarks.

Information Disclosure Statement

The Office Action states that reference 53 was not provided, and that references 54 and 56 did not include English language translations with the Information Disclosure Statement filed July 19, 2007. The Applicant hereby submits a copy of application EP 0717353 from the European Patent Office. Application EP 0717353 also published as Japanese Application JP8255104 (reference 53). Therefore, Applicant respectfully submits that an English language translation of JP8255104 has been submitted to the Office.

The Office Action also states that English language translations of references 54 (Japanese Application JP 11272454) and reference 56 (Japanese Application JP 11345127) were not provided. With respect to reference 54, Applicant respectfully submits that JP 11272454 also published as US Patent 6,199,204. With respect to reference 56 (JP 11345127), the Applicant hereby submits a copy of US patent 6,334,212, which claims priority from reference 56.

Rejections of Claims 1-24 Under 35 U.S.C. §103(a)

Claims 1-24 were rejected under 35 U.S.C. §103(a) as being unpatentable over Yang et al. (US Patent Ap. Pub.2003/0065738A1; hereinafter ("Yang") in view of Gauvin et al. (US 5,790,800; hereinafter "Gauvin"). The Applicant respectfully traverses the rejection.

The Applicant respectfully submits that the Examiner has failed to establish a case of prima facie obviousness for at least the reasons provided below. M.P.E.P. §2142 clearly states that "[t]he examiner bears the initial burden of factually supporting any prima facie conclusion of obviousness." The M.P.E.P. §2142 goes on to state that "[t]o establish a prima facie case of obviousness, three basic criteria must be met. First, there must be some suggestion or motivation, either in the references themselves or in the knowledge generally available to one of ordinary skill in the art, to modify the reference or to combine reference teachings. Second, there must be a reasonable expectation of success. Finally, the prior art reference (or references when combined) must teach or suggest all the claim limitations. The teaching or suggestion to make the claimed combination and the reasonable expectation of success must both be found in the prior art, and not based on applicant's disclosure."

Rejection of Claims 1-15

Independent claim 1 was rejected under 35 U.S.C. §103(a) as unpatentable over Yang in view of Gauvin. The Applicant respectfully traverses the rejection.

With regard to the rejection of independent claim 1, the Office Action concedes that Yang "fails to explicitly disclose: [ ] based upon a prior registration associating the one of the plurality of servers with the one of the plurality of software components making the at least one request for service." Page 4.

However, the Office Action states that Gauvin discloses in Fig. 1 and column 3, lines 56-58, and lines 65-67, what Yang fails to explicitly disclose. The Applicant respectfully disagrees.

Figure 1 describes "a block diagram of a distributed computer environment including a mobile client computer configured according to the principles of the

invention." Brief Description of the Drawings (column 3). It can be seen that Figure 1 does not disclose "a prior registration associating the one of the plurality of servers with the one of the plurality of software components making the at least one request for service."

Lines 56-58 and 65-67 in column 3 of Gauvin state: "The server computer 111-113 and the gateway or router 120 are made of standard hardware, and are configured to provide computing services to multiple client computers... the mobile computer 110 includes a communications manager (CM) 200 to provide connectivity between registered mobile client applications 210, and the fixed servers 111-113." The Applicant respectfully submits that this citation does not disclose "a prior registration associating the one of the plurality of servers with the one of the plurality of software components making the at least one request for service."

Rather, what Gauvin appears to disclose is that the "registered applications" are registered with a registry of a "standard operating system ... e.g., Microsoft Windows 3.1, and Windows95, NT, etc." Column 3, lines 44-47. This is based on the result of a search of Gauvin where the only places where the Applicant was able to find a form of the word "registration" were in column 4, lines 17-19, and column 5, lines 26-29.

For example, in column 4, lines 17-19, Gauvin states "The processes of the communications manager 200 are exposed to users of the registered applications 210 in two views: a set-up graphic users interface (GUI) 220, and a manager GUI 225." In column 5, lines 26-29, Gauvin states "This infrastructure can use windows message passing, and callback procedures of the interface 226 to notify the registered client applications 210 of connection events."

Accordingly, it can be seen that Gauvin does not disclose "registration associating the one of the plurality of servers with the one of the plurality of software components making the at least one request for service." If Gauvin specifically discloses "registration associating the one of the plurality of servers with the one of the plurality of software components making the at least one request for service," the Applicant respectfully requests a specific citation.

Based at least upon the above, the Applicant respectfully submits that the Office has failed to establish a prima facie case of obviousness, as required by M.P.E.P. §2142, and that the above rejection of claim 1 under 35 U.S.C. §103(a) cannot stand. The Applicant also respectfully submits that since claims 2-15 depend from claim 1, claims 2-15 are also allowable.

The Applicant respectfully requests, therefore, that the rejection under 35 U.S.C. §103(a) be withdrawn for claims 1-15.

Rejection of Claims 16-21

Independent claim 16 was rejected under 35 U.S.C. §103(a) as being unpatentable over Yang in view of Gauvin. The Applicant respectfully traverses the rejection.

With regard to the rejection of independent claim 16, the Office Action concedes that "Yang fails to explicitly disclose: [ ] based upon an association of the one of the plurality of service providers with the client-side component that made the request." Page 10.

However, the Office Action states that Gauvin discloses in Fig. 1 and column 3, lines 56-58, and lines 65-67, what Yang fails to explicitly disclose. The Applicant respectfully disagrees. The Applicant also notes that claim 16 is rejected for the same reason, using the same citations, as claim 1.

Accordingly, for at least the reasons stated above with respect to rejection of claim 1, the Applicant respectfully submits that Gauvin does not disclose "based upon an association of the one of the plurality of service providers with the client-side component that made the request." Therefore, the Applicant respectfully submits that the Office Action has failed to establish a prima facie case of obviousness, as required by M.P.E.P. §2142, and that the above rejection of claim 16 under 35 U.S.C. §103(a) cannot stand. Additionally, the Applicant respectfully submits that since claims 17-21 depend from claim 16, claims 17-21 are also allowable.

The Applicant respectfully requests, therefore, that the rejection under 35 U.S.C. §103(a) be withdrawn for claims 16-21.

Rejection of Claims 22-24

Independent claim 22 was rejected under 35 U.S.C. §103(a) as being unpatentable over Yang in view of Gauvin. The Applicant respectfully traverses the rejection.

With regard to the rejection of independent claim 22, the Office Action concedes that "Yang failed to explicitly disclose: registering at least one call-back function available in the software component; communicating, to the service broker, a request for updating of at least one of the software component and software component configuration; receiving results from a remote service provider; and invoking the at least one call-back function using the received results." Page 13.

The Office Action then states that Gauvin discloses in column 5, lines 24-30, "call back procedures to notify the registered client applications 210 of connection events. (FIG. 2, #226)," and proceeds to explain its reasoning for rejection of claim 22 on this basis. However, while the Applicant traverses this rejection, the Applicant has amended claim 22 for clarity in the interest of furthering prosecution.

Accordingly, for reasons similar to those put forth with respect to claims 1 and 16, the Applicant respectfully submits that Gauvin does not disclose "registering at least one call-back function available in the software component, wherein each of the at least one call-back function is associated with a server."

Based at least upon the above, the Applicant respectfully submits that the Office has failed to establish a prima facie case of obviousness, as required by M.P.E.P. §2142, and that the above rejection of claim 22 under 35 U.S.C. §103(a) cannot stand. The Applicant further submits that since claims 23-24 depend from claim 22, claims 23-24 are also allowable.

The Applicant respectfully requests, therefore, that the rejection under 35 U.S.C. §103(a) be withdrawn for claims 22-24.

Conclusion

In general, the Office Action makes various statements regarding claims 1-24 and the cited references that are now moot in light of the above. Thus, the Applicant will not address such statements at the present time. However, the Applicant expressly

reserves the right to challenge such statements in the future should the need arise (e.g., if such statements should become relevant by appearing in a rejection of any current or future claim).

The Applicant believes that all of pending claims 1-24 are in condition for allowance. Should the Examiner disagree or have any questions regarding this submission, the Applicant invites the Examiner to telephone the undersigned at (312) 775-8000.

A Notice of Allowability is courteously solicited.

Respectfully submitted,

Dated: November 30, 2007

/Kevin E. Borg/
Kevin E. Borg
Reg. No. 51,486

Hewlett-Packard Company
Intellectual Property Administration
Legal Department, M/S 35
P.O. Box 272400
Fort Collins, CO 80527-2400

(19)日本国特許庁（ＪＰ）　(12) 公開特許公報（A）　(11)特許出願公開番号

特開平8－255104

(43)公開日　平成8年(1996)10月1日

(51)Int.Cl.^{6}	識別記号	庁内整理番号	F I	技術表示箇所
G０６F　12/00	５１７	7623−5B	G０６F　12/00	５１７
	５１０	7623−5B		５１０B
9/06	４１０		9/06	４１０P

審査請求　未請求　請求項の数８　ＯＬ　（全 38 頁）

(21)出願番号　特願平7－325290

(22)出願日　平成７年(1995)12月14日

(31)優先権主張番号　０８／３５５８８９

(32)優先日　1994年12月14日

(33)優先権主張国　米国（ＵＳ）

(71)出願人　390035493
エイ・ティ・アンド・ティ・コーポレーション
AT&T　CORP.
アメリカ合衆国　10013－2412　ニューヨーク　ニューヨーク　アヴェニュー　オブ　ジ　アメリカズ　32

(72)発明者　ディヴィッド　ジェラルド　コーン
アメリカ合衆国　10003　ニューヨーク，ニューヨーク，エー107，マーサー　ストリート　303

(74)代理人　弁理士　岡部　正夫　（外２名）

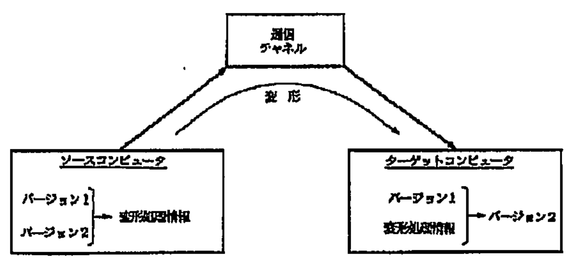
最終頁に続く

(54)【発明の名称】　ソフトウエアおよびデータの効率的かつ安全性の高い更新

(57)【要約】

【課題】　伝送データ量を削減し、かつかつ傍受されにくい、遠隔地からソフトウエア又はデータを更新する方法を提供する。

【解決手段】　本発明は、旧バージョンから新規バージョンへデータファイルを更新する装置に関する。本発明は、旧バージョンと新規バージョンを比較し、類似と相違に関する情報を含む変形処理情報を導き出す。さらに、本発明は、新規バージョンを導き出すために、新規バージョン自体を参照せずに、変形処理情報を用いて、旧バージョンの処理を行う。本発明を用いると、更新されたファイル自体を送信しなくても、変形処理情報を複数の場所に送信することにより、複数の場所にある銀行の記録などのような元のファイルの複数バージョンを更新することができる。一般に、変形処理情報には新規バージョンの内容全体が含まれていないことから、この方法は傍受に対して高い抗力を持つ。



【特許請求の範囲】

【請求項１】　コンピュータにおいて、

ａ）第二バージョンにアクセスしなくても、第一バージョンに基づいてバイナリファイルの第二バージョンの復元を可能にする命令を作成するプログラム手段から成る改良。

【請求項２】ａ）連結されたときに第二バージョンを復元する文字列を各々が作成する一連の命令を生成する段階と、

ｂ）各命令を実行する段階とから成る、第一バージョンから導き出されたバイナリファイルの第二バージョンを復元する方法。

【請求項３】デジタルコンピュータの場合、

ａ）

ｉ）第一および第二ファイルを検査し、

ｉｉ）第二ファイルを復元するため、実行されたときに、

Ａ）第一ファイルの一部と、

Ｂ）第二ファイルへのアクセスを行わずに第二ファイルの一部とを結合する一連の命令を作成する第一プログラム手段から成る改良。

【請求項４】　第二ファイルの復元を行うために一連の命令を実行する第二プログラム手段を具備する請求項３に記載の改良。

【請求項５】ａ）新規バージョンと旧バージョンとを比較し、かつ、

ｉ）新規バージョンが旧バージョンと似ている類似語句と、

ｉｉ）新規バージョンが旧バージョンと異なる相違語句を識別し、

ｂ）旧バージョンに各類似語句が出現しているアドレスを記憶し、

ｃ）新規バージョンに出現しているアドレスと共に各相違語句を記憶する段階とから成る、コンピュータファイルの新規バージョンを退避する方法。

【請求項６】ａ）第一サイトにおいて、第一ファイルと第二ファイルの

ｉ）類似と、

ｉｉ）相違

を検出する段階と、

ｂ）第二サイトにおいて、第一ファイルのコピーを保持する段階と、

ｃ）

ｉ）類似の位置と、

ｉｉ）相違自体と、

ｉｉｉ）（ｃ）（ｉ）、（ｃ）（ｉｉ）、および第二ファイルに基づく第二ファイルの復元を可能にする情報を、第二サイトに送信する段階とから成る情報の復元法。

【請求項７】ａ）

ｉ）新規バージョンの指定位置に旧バージョンの指定部分をコピーすることと、

ｉｉ）新規バージョンの指定位置に新規バージョンの指定部分をコピーすることと、

ｉｉｉ）新規バージョンの指定位置に指定バイトを加えることのうち、１つまたはそれ以上を行うよう指示する命令を受信する段階と、

ｂ）命令を実行する段階とから成る、ファイルの旧バージョンを新規バージョンへ更新する方法。

【請求項８】ａ）

ｉ）

Ａ）第一バージョンまたは第二バージョンの発信元位置から、

Ｂ）第二バージョンの宛先位置へ、文字がコピーされるよう指示するＣＯＰＹ命令と、

ｉｉ）指定文字が第二バージョンの指定位置に加えられるよう指示するＡＤＤ命令とから成る、並びを導き出す手段から成るファイルの第一バージョンと第二バージョンの相違を表すデータを作成するシステム。

【発明の詳細な説明】

【０００１】

【産業上の利用分野】本発明により用いられるソースコードを記載した付録を添付する。本発明は、伝送されるデータ量を削減し、かつ傍受されにくい、遠隔地からソフトウエアまたはデータもしくは両者を更新する方法に関する。

【０００２】

【従来の技術】電信電話会社の電話システムを介したコンピュータ間の電子データ通信には、スピードと機密保護が必要とされている。伝送前にデータを圧縮することにより伝送速度を向上できる一方で、暗号化により機密保護が可能である。データ圧縮は、データ源における冗長度を低下させ、かつ暗号解読への抗力を高めることから、暗号化にも有効な働きをする。しかし、データファイルが圧縮ならびに暗号化された場合、圧縮および暗号化されたデータは、単一のデータ源に基づくものとなる。単一のデータ源に基づくデータ圧縮および暗号化は、完全なものとはいえない。データの冗長度が高くなければ、圧縮はうまく機能せず、最新の暗号解読法と共に機能する高速コンピュータは、大抵の暗号化データを解読することができるからである。多くの場合、最初のデータ源に関する複数の新規バージョンが、データに若干の変更が加えられた後に頻繁に伝送される。この場合の変更が若干であるとは、全バージョンが本質的に類似していることを意味している。本質的に類似していれば、その類似性を利用して、あるバージョンを別のバージョンに変形する最小の変形処理情報を計算することが可能である。

【０００３】このような変形処理情報の計算方法は、データディファレンシング（データ間の差の計算）と呼ば

れている。データディファレンシングは伝達されるデータ量を削減する働きをすることから、これにより伝送速度が向上する。さらに、旧バージョンを持たない傍受者は、伝送されたデータからあまり多くの情報を引き出すことができないため、データディファレンシングは、プライバシーの保護にも役立つ。これまで数多くの圧縮およびディファレンシング技術が、検討されてきた。「情報理論に関するＩＥＥＥトランザクション（ＩＥＥＥ Ｔｒａｎｓａｃｔｉｏｎｓ ｏｎ Ｉｎｆｏｒｍａｔｉｏｎ Ｔｈｅｏｒｙ）」１９７７年5月号の２３（３）（３３７頁～３４３頁）に記載されたＪ．ＺｉｖならびにＡ．Ｌｅｍｐｅｌによる「順次データ圧縮のための普遍アルゴリズム（Ａ Ｕｎｉｖｅｒｓａｌ Ａｌｇｏｒｉｔｈｍ ｆｏｒ Ｓｅｑｕｅｎｔｉａｌ Ｄａｔａ Ｃｏｍｐｒｅｓｓｉｏｎ）」の記事では、単一のデータファイルを圧縮する技術について述べられている。この技術は、データの順序を分析し、かつ、可能であれば、各位置においてすでに分析された部分にある別のセグメントと一致する最長セグメントを識別することによって機能する。このようなセグメントが検出された場合、一致した位置と一致した長さを用いて暗号化が行われる。また、一致しなかったデータは、そのまま出力される。

【０００４】このアルゴリズムの具体化が、１９８４年8月7日にＷ．Ｌ． Ｅａｓｔｍａｎ、Ａ． Ｌｅｍｐｅｌ、およびＪ．Ｚｉｖ．に付与された米国特許４，４６４，６５０の主題であった。このＥａｓｔｍａｎ－Ｌｅｍｐｅｌ－Ｚｉｖ圧縮方法では、圧縮解除中も圧縮中とほぼ同じ作業量をこなさなければならないことから、圧縮解除の際に低速となる。さらに、この方法は、データディファレンシングに応用できない。ＵＮＩＸシステムでは、第一ファイルから第二ファイルへの変形の際に削除または追加する必要のある一連の行を作成する「ｄｉｆｆ」プログラムを用いて、（バイナリ以外の）テキストファイル間の違いを算出することができる。「ｄｉｆｆ」による方法を採用した場合、テキスト行の若干の変更によって、極めて大幅な変形は別のものとみなされることから、そうした大幅な変形を行う可能性も出てくる。さらに、この方法は、テキストファイルでしか機能しないことから、応用範囲が限られている。「コンピュータシステムに関するＡＣＭトランザクション（ＡＣＭ Ｔｒａｎｓａｃｔｉｏｎｓ ｏｎ Ｃｏｍｐｕｔｅｒ Ｓｙｓｔｅｍｓ）」１９８４年１１月号の２（４）（３０９頁～３２１頁）に記載されたＷａｌｔｅｒ Ｆ． Ｔｉｃｈｙによる「ブロックの移動によるストリングからストリングへの訂正上の問題（Ｔｈｅ Ｓｔｒｉｎｇ－ｔｏ－Ｓｔｒｉｎｇ Ｃｏｒｒｅｃｔｉｏｎ Ｐｒｏｂｌｅｍ ｗｉｔｈ Ｂｌｏｃｋ Ｍｏｖｅｓ）」の記事では、データファイルのあるバージョンから別のバージョンへ変形を行うブロック移動と呼ばれる一連の命令を計算するアルゴリズムについて述べられている。ブロック移動（ｂｌｏｃｋ－ｍｏｖｅ）は、第一バージョンの別のセグメントと一致する第二バージョンのデータセグメントである。このアルゴリズムは、本質的に実現されないものであり、第二バージョンに固有の冗長性がある場合に、それを活用して第一バージョンから第二バージョンを作成するのに必要な変形を最小限度に抑えることができない。

【０００５】

【発明の概要】本発明の一態様において、コンピュータプログラムにより、ファイルの第一バージョンと第二すなわち新規バージョンを比較し、一連の命令を作成する。この命令により、第一バージョンに基づいた第二バージョンの復元が可能になる。上記の命令は、２つのタイプに分けられる。ひとつはＣＯＰＹ命令であり、指定された一連の文字を構成中のファイルにコピーするよう命令する。指定された文字列は、第一バージョンまたは構成中のファイルのいずれにも存在可能である。もうひとつは、ＡＤＤ命令であり、このＡＤＤ命令に伴う一連の文字の追加を指示する。この２種類の命令が連続的に実行されると、コピーならびに追加された各文字列の連結が行われ、第二バージョンの復元が行われる。

【０００６】

【実施例】以下に、簡単な類比により、本発明のより基本的な側面を一部説明する。また、本発明の一部を実現するコンピュータコードについての技術的な説明は、「本発明について」と題した項目にて行う。

類推

ある新聞記者と新聞編集者が、記事を一緒に、しかし、異なった場所で書くものとする。さらに、両者が図１Ａに示すバージョン１を書いたと仮定する。また、この新聞記者が図１Ｂに示す変更を行った結果、図１Ｃ（四角枠は変更した位置を示している）に示すようなバージョン２となり、このバージョンを編集者に送信したいと考えていると仮定する。そこで、次の手順を用いれば、バージョン２の内容全体を送信せずに、編集者にバージョン２の送信が可能である。

【０００７】手順

まず初めに、記者は、図１Ａに示すように、バージョン１の各語に位置を割り当てる。５０語に対し、５０箇所の位置がある。次に、記者は、バージョン１の単語表を確認する。単語表は、図１Ａに示すような使用される語のリストすなわち一覧表である。ただし、各語は、単語表に一度だけしか現れない。例えば、位置１８の語「ｔｈｅｍ」は単語表に記入されない。この語がすでに位置１６にあるからである。各語が一度しか記入されていないことを確認するために、記者は、各単語を入力する度に、その語が表にあるかどうか見ながらチェックする。例えば、記者が、最初の単語「Ｗｈｅｎ」を入力する。次に、位置２の語「ｂｕｙｉｎｇ」を取り上げ、単語表の最初の単語、すなわち、「Ｗｈｅｎ」と照合する。一

致しなければ、記者は、「ｂｕｙｉｎｇ」を単語表の位置２に記入する。記者は、「ｔｈｅｍ」という語が位置１８にくるまでこのようにして処理を進める。記者がこの語を単語表内にあるそれまでに入力された１７の項目と照合してみると、「ｔｈｅｍ」が単語表の位置１６にある「ｔｈｅｍ」と一致することに気がつく。したがって、後の「ｔｈｅｍ」は、前の「ｔｈｅｍ」と重複するため単語表に記入されない。

【０００８】バージョン１の最後の語について一致の確認が行われると、単語表は完了する。バージョン１には、全体の長さに対し５０語含まれているが、単語表の長さが示す通り、異なる語は３８語しかない。次に、記者は、図１Ｃにあるように、バージョン２の単語表を作成する。バージョン２には、新語として示されている追加の３語を除き、バージョン１と同じ単語表がある。これで、記者は、編集者にバージョン２を送信する準備が整ったことになる。記者は、次の６つのメッセージすなわち命令を送信してバージョン２を伝送する。各命令は、図１Ｄ～図１Ｉの各図によって理解できる。

１．　ＣＯＰＹ　２　１
２．　ＡＤＤ　１　Ｈａｗａｉｉａｎ
３．　ＣＯＰＹ　３　３
４．　ＡＤＤ　１　ａｌｌ
５．　ＣＯＰＹ　３７　８
６．　ＡＤＤ　２　ａ　ｂｒｉｃｋ

【０００９】命令１
事前配列により、記者と編集者は、図１Ｄに示すように、位置１にポインタを設定する。最初の命令「ＣＯＰＹ　２　１」は、「バージョン１の位置１から始まる２語の並びをコピーせよ」を意味している。（各命令では、該当する語がバージョン２のポインタの位置に配置されることが暗黙の了解となっている。）この動作が、図１Ｄに示されている。ただし、構文は、以下の通りである。
ＣＯＰＹ［語数、バージョン１の開始位置］
命令を実行した後、編集者は、ＣＯＰＹ命令にある「語数」（この場合、２語）分だけポインタを移動させる。これにより、ポインタは、図１Ｅに示す位置に設定される。

【００１０】命令２
命令２では「ＡＤＤ　１　Ｈａｗａｉｉａｎ」とあり、「’Ｈａｗａｉｉａｎ’という１語を付け加えよ」を意味している。この動作は、図１Ｅに示されている。ただし、構文は以下の通りである。
ＡＤＤ［語数、該当する語］
命令を実行した後、編集者は、ＡＤＤ命令にある「語数」（この場合、１語）分だけポインタを移動させる。これにより、ポインタは、図１Ｆに示す位置に設定される。

【００１１】命令３
命令３の「ＣＯＰＹ　３　３」は、「バージョン１の位置３から始まる３語の並びをコピーせよ」を意味している。この動作は、図１Ｆに示されている。ただし、構文は命令１と同一である。命令を実行した後、編集者は、ＣＯＰＹ命令にある「語数」（この場合、３語）分だけポインタを移動させる。これにより、ポインタは、図１Ｇに示す位置に設定される。

【００１２】命令４
命令４では「ＡＤＤ　１　ａｌｌ」とあり、「’ａｌｌ’という１語を付け加えよ」を意味している。この動作は、図１Ｇに示されている。命令を実行した後、編集者は、ＡＤＤ命令にある「語数」（この場合、１語）分だけポインタを移動させる。これにより、ポインタは、図１Ｈに示す位置に設定される。

【００１３】命令５
命令５の「ＣＯＰＹ　３７　８」は、「バージョン１の位置８から始まる３７語の並びをコピーせよ」を意味している。この動作は、図１Ｈに示されている。ただし、構文は命令１と同一である。命令を実行した後、編集者は、ＣＯＰＹ命令にある「語数」（この場合、３７語）分だけポインタを移動させる。これにより、ポインタは、図１Ｉに示す位置に設定される。

【００１４】命令６
命令６では「ＡＤＤ　２　ａ　ｂｒｉｃｋ」とあり、「’ａ　ｂｒｉｃｋ’という２語を付け加えよ」を意味している。この動作は、図１Ｉに示されている。これで、バージョン２が作成されたことになる。

【００１５】重要な特徴
図１Ｃに示されるバージョン２には、４３語が含まれており、語自体は、１６３文字から構成されていた。しかし、６つの命令には２７文字プラス命令自体（ＣＯＰＹとＡＤＤ）が含まれていた。各命令と各文字を１バイトとしてコード化した場合、全体のメッセージは、２７＋６文字、すなわち、３３文字となる。３３文字を送信すれば、１６３文字を送信した場合に比べて大幅に時間が短縮される。

【００１６】本発明について
上記の類比の説明は簡単に述べたものであり、本発明の特徴のすべてを示しているわけではない。図２では、ファイルに関する２つのバージョンの他に、バージョン１と組み合わせてバージョン２を作成できるようにする「変形処理情報」を示している。図３の手続きでは、復元が行われる。この手続きについてこれから説明する。まず初めに、一般的なパターンが綿密に作成される。予備知識として、重要点を４つ認識しておく必要がある。ひとつは、各バージョンの各文字は、図２Ａに示す通り、番号付けされた位置を占めている。例えば、バージョン１の場合、最初の「ａ」は位置０を占有している。最初の「ｂ」は位置１を占有している等である。第二に、バージョン２の番号付けされた位置は、バージョン

１の番号付けされた位置の中で最も高い数字の位置の後ろから開始する。したがって、バージョン１の最も数字の高い位置が「１５」であることから、バージョン２は「１６」から開始することになる。第三に、ポインタ（図３の手続き中にある変数）は、図２Ａに示すような現在位置を示している。第四に、「変形処理情報」は、複数の命令により構成されている。命令には２種類、すなわちＡＤＤとＣＯＰＹがある。次に、この２種類の命令の動作について、バージョン１からバージョン２がどのように構成されるのかを示しながら説明したい。

【００１７】命令１

図２Ｂでは、「変形処理情報」の命令１が実行されている。命令「ＣＯＰＹ４　０」は、基本的に、「バージョン１の位置０から始まる４文字をバージョン２のポインタの位置にコピーせよ」を示している。（下線の語は、命令内の語を示している。）この動作は、図２Ｂに示されている。上記の命令では、バージョン２の作成に使用される文字を、命令自体からではなく、バージョン１から取得している。このため、電話送信により命令を取得する場合でも、傍受者は、バージョン１にアクセスできないと推定されるため、バージョン２の作成に使用できる情報は一切得ることができない。構文は、以下の通りである。

ＣＯＰＹ［語数、開始位置］

【００１８】命令２

図２Ｃでは、命令２が実行される。命令「ＡＤＤ　２ｘ，ｙ」は、事実上、「ｘとｙの２文字をポインタの位置を起点にバージョン２に追加せよ」を意味している。この命令では、バージョン２の作成に用いられる文字を命令自体から得ている。その結果、傍受者は、バージョン２に関する情報の一部を取得できる。しかし、実際には、この種の命令は、非常に多種多様な（情報を一切含まない）ＣＯＰＹ命令と混合して用いられることが予想されるため、傍受者は、バージョン２について重要な情報を得ることはない。にもかかわらず、理論的には、ＡＤＤ命令のみが「変形処理情報」に含まれる場合もあり得ることであり、その場合、傍受者は、バージョン２をそのまま取得することになる。あるいは、ＡＤＤ命令の数が膨大になることもある。傍受者がこうしたＡＤＤ命令から情報を取得できないようにするために、暗号化オプションが提供されており、内容については後に述べる。ＡＤＤ命令の構文は以下の通りである。

ＡＤＤ［語数、該当文字］

【００１９】命令３

図２Ｄでは、命令３が実行されている。命令「ＣＯＰＹ　６　２０」は、事実上、次の内容を示している。位置２０とポインタとの間にある文字を用いて、長さが６文字のコピーを作成し、ポインタの位置にコピーした文字を配置せよ。命令を実行した結果、図示されるように、ｘｙｘｙｘｙとなる。（別の例として、ＣＯＰＹ文で「ＣＯＰＹ　６　１９」としてあれば、事実上、次の内容を意味している。位置１９（２０ではない）とポインタとの間にある文字を用いて、長さが６文字のコピーを作成し、ポインタの位置にコピーした文字を配置せよ。この場合、位置１９とポインタとの間にある文字列はｄｘｙであることから、ポインタの位置に付け加えられた文字の並びはｄｘｙｄｘｙとなる。）

この命令の場合、バージョン２の作成に用いられる文字がバージョン２から取得される点が重要である。したがって、ＣＯＰＹ命令は指定されたアドレスによって２つのデータ源を使用していることがわかる。この命令では、アドレス（すなわち、「ＣＯＰＹ　６　２０」の「２０」）が、バージョン２を示している（すなわち、アドレスは１５を上回る）。したがって、バージョン２が、このコマンドのデータ源である。逆に、アドレスが１５以下であれば、バージョン１がデータ源として用いられるはずである。この方法を用いると、事実上、ＡＤＤ命令を使用して単語表を拡張することができる。したがって、（ａ）バージョン１の中になく、（ｂ）拡張された単語表にある（以前に追加された）文字がバージョン２に含まれていることが明らかになった場合、ＣＯＰＹ命令が使用できる。したがって、この場合のＣＯＰＹ命令には２つの利点がある。ひとつは、ＣＯＰＹ命令では、文字の総数と開始位置を示すだけで大量の文字を挿入することができる。逆に、ＡＤＤ命令を用いた場合、送信される文字自体を必要とすることから、より長い送信が必要となる。もうひとつは、上記の通り、命令を傍受した者がバージョン１を取得するようなことがない限り、ＣＯＰＹ命令には一切の情報が含まれていないことになる。

命令４

図２Ｅでは、命令４が実行されている。命令「ＣＯＰＹ　５　９」は、事実上、「バージョン１の位置９から始まる５文字をコピーせよ」を示している。この命令は、命令１とよく似ている。

【００２０】図３のプロセスに関する参照事項

図３は、コンピュータがバージョン１と「変形処理情報」との組み合わせからバージョン２を作成するプロセスを示している。図３では、２行目において、図２Ｂ～図２Ｅに示されているポインタの計算に用いられる変数ｃの初期化が行われている。ポインタの計算は、９行目、１３行目、および１４行目で適宜行われている。

命令１

図２に示される「変形処理情報」の命令１（ＣＯＰＹ　４　０）は、図３の１３行目で実行されている。変数ｐは、原始データの開始位置を示しており、命令の中から得ることができ、この場合、０である。ｐはｎ（１２行目）よりも小さいことから、データ源はバージョン１にある。その結果、ＩＦ文により１３行目が実行されると、バージョン１からデータがコピーされる。このと

き、バージョン１＋ｐの位置、すなわち位置０から開始する。変数ｓは、文字数を表しており、７行目の命令から得ることができ、この場合、４である。ポインタは、１６行目において更新される。

命令２

命令２（ＡＤＤ　２　ｘ，ｙ）は、８行目のＩＦ文により、９行目で実行される。これにより、バージョン２＋ｃの位置にある長さｓ（２に等しい）の文字列が設定される。ポインタは、１６行目で更新される。

命令３

命令３（ＣＯＰＹ　６　２０）は、１２行目[ｐ（命令から得られ、２０に等しい）はｎより小さい]により１４行目において実行される。このコピー関数は、データ源としてバージョン２を用い、バージョン２＋ｃから開始する。コピー関数は、文字数ｓの長さを有する並びである。（変数ｓは[ポインター２０]により得られ、２０の数字は命令から得られる。）

命令４

命令４は、命令１のように１３行目において実行される。

【００２１】「変形処理情報」の作成

図５は、各バージョンを１文字ずつ確認し、かつバージョン１と変形処理情報と呼ばれる命令との組み合わせからバージョン２を構成できるようにする一連の命令を作成する手続きを示している。図２の変形処理情報については、すでに説明している。この手続きによって採られる一般的な手法について、以下に説明する。

実行１

まず初めに、バージョン１の処理が行われる。バージョン１は、図２Ｆの数箇所に表示されている。図５のコードでは、位置０から始まる４文字の文字列が前の位置から始まる４文字の文字列と一致するかどうか尋ねる。位置０の前には何も存在しないことから、当然、答えは「Ｎｏ」である。したがって、位置０は、「実行１」と表示された列に示されるように、カラットを用いてフラグが立てられている。

実行２

実行２では、コードによって、似たような質問、すなわち、位置１から始まる４文字の文字列が前の位置から始まる４文字の文字列と一致するかどうかを尋ねる。答えは「Ｎｏ」であり、位置１にフラグが立てられる。

実行３および実行４

同様に、フラグが位置２および位置３に設定され、図２Ｆの実行４のようにフラグが立てられる。

【００２２】実行５

実行５では、上記とは違った結果が得られる。実行５では、コードにより通常の「位置４から始まる４文字の文字列は、前の位置から始まる４文字の文字列と一致するか？」という問いを出す。実行５として示された列からわかるように、答えは、位置０において、「Ｙｅｓ」である。位置４にはフラグが表示されず、ＥＸＴＥＮＤ関数（図５の１６行目）が呼び出されて動作が行われる。ＥＸＴＥＮＤ関数は、一致した文字列の長さを尋ねる。一致したブロックの後ろの次の位置（一致したブロックは位置４～７を占めていることから、次の位置は、位置８である）に４つ前の位置と同じものがあるかどうか尋ねる。答えは、試行ＴＡからわかるように、「Ｙｅｓ」である。次に、一致したブロックの２番目の位置すなわち位置９に、４つ前の位置と同じものがあるかどうか尋ねる。試行ＴＢからわかるように、今度の答えも「Ｙｅｓ」である。この問いは、そのような一致が検出されなくなる試行ＴＥに位置が到達するまで続行される。したがって、ＥＸＴＥＮＤ関数は、位置４～位置１１までの８つの位置にわたり一致が見られることを確認した。コードの論理上、拡張された一致の最後の３つの位置、すなわち、「実行５の結果」の中の「ＢＣＤ」にフラグが立てられる。

【００２３】実行６

実行６は、位置１２（最も右にあるｅ）から始まる４文字の並びが前に発生した位置から始まる４文字の文字列と一致しているかどうか尋ねる。ここでは、答えは「Ｎｏ」であり、「実行６」と表示された列に示されるように、位置１２にフラグが立てられる。

【００２４】結果

図２Ｆの下部に結果が示されており、「結果」と表示されている。そこで、重要な特徴について２点以下に説明する。第一に、後に説明するように、フラグは、バージョン２の作成に用いられる並びの開始だけでなく、終了も示している。第二に、図示されるように、フラグが立てられない領域がある。バージョン２の作成に用いられる文字列の探索は、こうした領域の外から開始してその領域に侵入することができるが、このような領域内から開始することはない。したがって、探索開始点が削除されることから、全体の探索時間は短縮される。

【００２５】バージョン２の処理

実行１～実行４

次に、更新バージョンであるバージョン２の処理が行われる。コードにより、図２Ｇに示されている位置１６、１７、１８、および１９から始まる４文字の並びが以前の位置から始まるものと一致しているかどうか質問される。位置１６から始まる４文字の並びが位置０から始まるものと一致しているが、後続の３つが一致していないため、図２Ｇの上部に示されているように、位置１７、１８、および１９にフラグが立てられる。ＥＸＴＥＮＤ関数では、拡張された一致が全く検出されない。図５のコードは、２９行目にジャンプして、図２に出てくる上記の「ＣＯＰＹ　４　０」命令を発行する。全体の結果は、「出力」と表示された矢印で示されている。すでに３文字にフラグが立てられており、ＣＯＰＹ命令が発行されている。ただし、これらのフラグが立てられた文字

は、バージョン１のフラグが立てられた文字がそうであったように、後に、探索初期設定点として扱われることに注意しなければならない。

【００２６】実行５

図５のコードでは、（図２Ｇの「実行５」と表示されている欄のバージョン２にある）位置２０から始まる４文字の並び「ｘｙｘｙ」が前の位置から始まるものと一致しているかどうか尋ねる。コードにより、現在バージョン２に存在するものも含め、各フラグから探索を開始する。さらに、各フラグにおいて、コードによって前後両方向に探索が行われる。（この前後方向の探索については、簡潔化を図るため、これまでの説明では述べられていない。）全体の探索は、図２Ｇの実行５によって示されている。探索は、（ａ）各フラグから開始し、（ｂ）４文字を前後両方向にサーチすることから、試行Ｔ１～試行Ｔ１２のすべての４文字の並びが検査される。さらに、バージョン２のフラグを立てられた位置、すなわち、位置１７、１８、１９、および２０から、同じような探索が行われる。一致が検出されなかったことから、位置２０にあるｘにフラグが立てられる。

【００２７】実行６

図５のコードでは、（バージョン２にある）位置２１から始まる４文字の並び「ｙｘｙｘ」が前の位置から始まるものと一致しているかどうか尋ねる。位置２０については、バージョン２に存在するものも含め、各フラグから前後両方向にコード探索を開始する。現在のフラグの状態は、図２Ｈの右上の部分に示されている。

【００２８】実行７

図５のコードでは、（バージョン２にある）位置２２から始まる４文字の並び「ｘｙｘｙ」が前の位置から始まるものと一致しているかどうか尋ねる。答えは、位置２０において「Ｙｅｓ」である。このため、図２Ｈに示されるように、位置２２にフラグは立てられず、ＥＸＴＥＮＤ関数（図５の１６行目）が入力される。このＥＸＴＥＮＤ関数では、図２Ｆに関して述べられた方法により、一致が図２Ｈの位置２７まで達していると判断される。したがって、拡張された一致の最後の３つの位置にフラグが立てられ、図の左下の四角枠に示されているような結果となる。このとき、論理上、図５の２９行目に進み、２つの命令「ＡＤＤ　２　ｘ，ｙ」および「ＣＯＰＹ　６　２０」が発行される。（２９行目の引き数「ａｄｄ」および「ｃ」は、ＡＤＤ命令に関するものであり、引き数「ｐｏｓ」および「ｌｅｎ」は、ＣＯＰＹ命令に関するものである。）

【００２９】実行８

コードでは、位置２８から始まる文字の並び「ｂｃｄｅ」が前の位置から始まるものと一致しているかどうか尋ねる。答えは、バージョン１（図２Ｇの左上を参照）の位置９において「Ｙｅｓ」である。これで、ＥＸＴＥＮＤ関数が呼び出され、一致が追加文字「ｆ」まで延びていると判断する。次に、コードにより、２９行目において、命令「ＣＯＰＹ　５　９」を発行する。これで、図２の４つの命令が作成されたことになる。したがって、バージョン１プラス命令からバージョン２を復元できる。

【００３０】特色

前記の概要から、次の原理を読み取ることができる。第一に、ある位置が前に一致した並びを表しているとの判断が下されると、その位置にはフラグが立てられず、その結果、冗長であるとの理由により、該当する位置において後続の探索は開始されない。このため、（可能な場合）探索位置が削減される。第二に、バージョン２は、２種類のデータ源、すなわち、（ａ）バージョン１またはバージョン２からコピーされた文字列、（ｂ）バージョン２に追加された文字列により構成される。さらに、追加された文字列は、後でコピー動作に使用できる。別の観点から、第一ユーザがバージョン１を保有している場合、また、第二ユーザがバージョン１とバージョン２の両方を保有している場合、第二ユーザが以下の文字列を識別していれば、第一ユーザは、バージョン２の複製を作成できる。

（ａ）第一ユーザのバージョン１からコピーしたもの

（ｂ）第一ユーザのバージョン１に付け加えられたもの

（ｃ）第一ユーザのバージョン２の複製からコピーされたもの

【００３１】第三に、各フラグ表示された位置が、単語表内の１語の開始点を表している。これに類似する語が図１Ａに示されている。しかし、図１Ａの用語範囲とは異なり、各フラグ表示された位置が表す用語範囲は、極めて多くの並びを表現の対象としている。例えば、バージョン１が以下の通りであるとする。

ａｂｃｄｅＦｇｈｉｊｋ

大文字のＦ（位置６）にフラグが立てられた場合、次の並びを表している。

ｆ　ｆｇ　ｆｇｈ　ｆｇｈｉ　ｆｇｈｉｊ　ｆｇｈｉｊｋ

上記の並びは、他のフラグ表示された位置を含むことがある。したがって、単一のフラグ表示された位置は、バージョン２へのコピー対象となる多数の並びを表すこともあり得る。このため、このようなフラグ表示された位置が多数のＣＯＰＹ命令に現れることもある。例えば、「ＣＯＰＹ　３　６」は、「ｆｇｈ」をコピーすることを意味しており、また、「ＣＯＰＹ　５」は、「ｆｇｈｉｊ」をコピーすることを意味している。

【００３２】機密保護

すでに述べた通り、ＡＤＤ命令には、バージョン２の内容に関する情報が含まれており、このような情報には、機密保護が必要である。機密保護の１手法が図１Ｊに示されている。バージョン１内にポインタがくるように、送信された図２のＡＤＤ命令（「ＡＤＤ　２　ｘ，

y」）が修正される。修正された命令は、図1Jに「送信命令」と表示されている。この場合、ポインタが「2」であり、「c」を示している。次に、送信された情報（すなわち、「x」および「y」を表すバイト）が、「c」で始まるデータとの排他的論理和がとられる。図1Jの左側は、排他的論理和の動作を示している。送信される命令は、「ADD　2　2」プラス排他的論理和の動作結果である。傍受者はバージョン1に全くアクセスできないことから、この排他的論理和の結果には、傍受者にとって価値のある情報は一切含まれていない。

【0033】図1Kの右側にある受信された命令に含まれるデータは、バージョン1内の同じデータ、すなわち、「c」で始まるデータとの排他的論理和がとられる。この排他的論理和により、元のデータ、すなわち、「x，y」が回復する。この手続きでは、排他的論理和の動作の特性、すなわち、第一の語と第二の語の排他的論理和をとることにより第三の語を作成する点を利用している。第三の語を第二の語と排他的論理和をとることにより、第一の語を回復できる。このため、以下の送信された命令から、

ADD　2　2［EX−ORの結果］

目的とする命令である次の命令を得ることができる。

ADD　2　x，y

【0034】<u>重要事項</u>

1．COPYおよびADD命令が現れる順序は、当然、重要である。その順序が全く変われば、異なったバージョン2が得られる。別の観点からすれば、文字列自体は一種の情報である。文字列は、命令の組み合わせとして見ることもできる。英語のアルファベットの並べ換えによりワードが生成され情報が伝達されるように、このような並べ換えの仕方に情報が含まれている。文字列の復元を可能にする情報が含まれていれば、当然、順序をバラバラにして送信することもできる。例えば、各命令に番号付けをしてもよい。命令の正しい並びがどのように実行されるかという点とは無関係に、実行により、連結プロセスによってバージョン2の複製が作成される。すなわち、図2B〜図2Eの例に戻り、

1．最初の「abcd」が複製（図2B）に書き込まれる。「abcd」は、バージョン1から得られたものである。

2．次に、「xy」が連結される（図2C）。「xy」は、バージョン1から得られたものである。

3．次に、「xyxyxy」の連結が行われる（図2D）。「xyxyxy」は、バージョン2から得られたものである。（代わりに、4組の「xy」をバージョン1から取得し、ステップ2で連結することもできる。しかし、これではあまり効率的とはいえない。）

4．「bcdef」の連結が行われる（図2E）。「bcdef」は、バージョン1から得たものである。

【0035】2．さらに、上記の重要事項1に関し、各COPYおよびADD命令にはコピーまたは追加された部分の長さが含まれていることを発明者から指摘しておく。このため、所定のCOPYまたはADD命令について、コピーや追加を行うバージョン2内のアドレスを、以前の全体の長さに基づいて直ちに計算することができる。（このような長さに基づきポインタの計算が行われていることから、図3のコードにこの状態が示されている。）

【0036】3．本発明は、任意の種類のデータファイルを更新する際に使用でき、テキストファイル等の特定の種類のファイルに限定されるものではない。本発明は、一般に、バイナリファイルを取り扱うことができる。ファイルには文字が含まれている。各文字は、通常、1バイトのデータによって表される。1バイトに8ビットが含まれていることから、28、すなわち、256の想定可能な文字が1バイトで表現できる。「テキスト」ファイルでは、このような想定可能な組み合わせをすべて使用するわけではなく、英数字および句読点を表すものだけを使用する。「バイナリ」ファイルでは、想定可能な256の組み合わせがすべて使用される。本発明では、バイナリファイルの取り扱いが可能である。これとは対照的に、従来技術のプログラムのディファレンシングでは、テキストファイルしか処理できない。

【0037】4．本発明は、記憶されているバージョン1からバージョン2の遠隔地への復元動作に限られるものではない。さらに、プログラムの新規バージョンは、バージョン全体を記憶するのではなく、ADDおよびCOPY命令を用いて一箇所に記憶することができる。この方法により、記憶スペースが節約される。バージョンの復元を必要とする場合、図3のプログラムが実行される。

5．復元されるファイルのバージョンが、その時点において年代的に早いファイルの後のバージョンである必要はない。例えば、バージョン1は、バージョン2から復元できる。

【0038】<u>技術上の説明</u>

図3および図5に示されているコードについて、より技術的な説明を行う。

<u>概要</u>

データファイルは、バイトの並びとみなすことができる。実質的にはすべての現行のコンピュータ上において、1バイトは、記憶、通信、およびメモリ内のデータの操作によって効率的に圧縮できる最小の自然単位である。「データファイル」という語は、ディスク上に記憶されたファイルを指す場合が多いが、本発明では、そのようなバイトの並びはメインメモリのセグメントにすることも可能である。図1では、本発明を用いて2台のコンピュータ間でデータを同期化している例を示している。第一に、ソースコンピュータ（翻訳用計算機）上で

は、データの２つのバージョンであるバージョン１およびバージョン２が比較されて、バージョン１をバージョン２に取り込む変形処理情報を作成する。この変形処理情報は、何らかの通信チャネルを介してターゲットコンピュータ（目的計算機）に送信される。次に、ターゲットコンピュータ上では、変形処理情報とバージョン１のローカルコピーを用いてバージョン２を復元する。

【００３９】変形処理命令のコーディングとデコーディング

本発明により計算された変形処理情報は、２種類の命令、すなわち、ＣＯＰＹおよびＡＤＤの並びにより構成されている。バージョン２の復元中は、ＣＯＰＹ命令によりコピー対象となるデータの現存するセグメントの位置と長さが定義され、ＡＤＤ命令により追加対象となるデータのセグメントが定義される。図２では、バージョン１が「ａｂｃ　ｄａｂ　ｃｄａ　ｂｃｄ　ｅｆｇｈ」のバイトの並びにより構成されている２つのデータファイルの例を示している（ここでは読みやすくするためにスペースを挿入している）。バージョン２は、「ａｂｃ　ｄｘｙ　ｘｙｘ　ｙｘｙ　ｂｃｄｅｆ」の並びにより構成されている。したがって、バージョン１の長さは１６、バージョン２の長さは１７となっている。図２に示されている変形処理情報では、バージョン１からバージョン２を構成し直すうえで必要となる命令が示されている。ただし、バージョン１の位置は０からコード化され、バージョン２の位置はバージョン１の長さからコード化されるという規約を採用している。例えば、図２の変形処理情報の第三の命令は、２０としてコード化されたバージョン２の位置４からの６バイトをコピーするＣＯＰＹ命令である。

【００４０】図３では、一般にバージョンの復元が行われる手続きが示されている。１行目では、変数「ｎ」をバージョン１の長さに初期化する。２行目では、バージョン２の現在位置「ｃ」を０に設定する。３行目では、５行目でエンドオブファイル状態が検出された後に６行目で終了するループを開始する。４行目では、関数ｒｅａｄｉｎｓｔ（）を呼び出して命令を読み取る。７行目では、関数ｒｅａｄｓｉｚｅ（）を用いてコピーするサイズすなわちデータサイズを読み取る。８行目と９行目では、命令がＡＤＤ命令であるかどうか確認し、そうであれば、現在位置ｃから開始するバージョン２にデータを読み込む。１０行目～１５行目では、位置コードで読み取りを行って、バージョン１またはバージョン２から適宜コピー動作を行うことにより、ＣＯＰＹ命令の処理を行う。１６行目では、バージョン２の現在のコピー位置を新たに復元されたデータの長さ分だけ増加させる。ｃｏｐｙ（）関数は、ディスクメモリ（またはメインメモリ）の１領域から別の領域へデータをコピーする単純関数である。しかし、ｒｅａｄｉｎｓｔ（）、ｒｅａｄｓｉｚｅ（）、ｒｅａｄｐｏｓ（）、およびｒｅａｄｄａｔａ（）関数は、ＣＯＰＹおよびＡＤＤ命令とそのパラメタがどのようにコード化されるかという点についての具体的な定義に基づいて定義されなければならない。

【００４１】図３の手続きを図１の例に当てはめてみると、復号化には４つのステップがあることがわかる。第一ステップでは、位置０から始まるバージョン１から「ａｂｃｄ」の４バイトをコピーする。第二ステップでは、「ｘｙ」の２データバイトを追加する。第三ステップでは、（０から数えて規約により２０としてコード化された）位置４から始まる６バイトをバージョン２からコピーする。ただし、このステップの開始時点では、「ｘｙ」の２バイトしかコピーに使用できないことから、バージョン２の最初の６バイトである「ａｂｃｄｘｙ」が復元されただけである。しかし、データが左から右へコピーされることから、１バイトがコピーされるときは必ず、作成されているはずである。第四および最終ステップでは、バージョン１の位置９から「ｂｃｄｅｆ」の５バイトがコピーされる。

【００４２】以上で、ＣＯＰＹおよびＡＤＤ命令のコード化について説明がなされたことになる。このような特種な具体例である上記命令が選択されたのは、発明者の実験により、多くの異なるタイプのデータに対してこのような命令がうまく機能するためである。以上の説明がなされれば、上記のｒｅａｄｉｎｓｔ（）、ｒｅａｄｓｉｚｅ（）、およびｒｅａｄｐｏｓ（）機能は、容易に実行できる。各命令は、制御バイトから開始してコード化が行われる。制御バイトの８ビットは２つの部分に分けられている。最初の４ビットは０～１５の数を表しており、各々は、命令の種類と何らかの補助情報に関するコーディングを定義している。以下に、最初の４ビットに関する最初の１０の値の一覧を示している。

０：　ＡＤＤ命令

１、２、３：　ＱＵＩＣＫキャッシュの位置を伴うＣＯＰＹ命令

４：　ＳＥＬＦとしてコード化された位置を伴うＣＯＰＹ命令

５：　ＨＥＲＥとの差としてコード化された位置を伴うＣＯＰＹ命令

６、７、８、９：　ＲＥＣＥＮＴキャッシュからコード化された位置を伴うＣＯＰＹ命令

ＱＵＩＣＫキャッシュは、サイズ７６８（３ｘ３５６）の配列である。この配列の各指標には、「ｐ　ｍｏｄｕｌｏ　７６８」が配列の指標となっているように、新たなＣＯＰＹ命令の位置の値ｐが含まれている。このキャッシュは、各ＣＯＰＹ命令が（コーディング中に）出力されるか、または、（デコーディング中に）処理された後、更新される。タイプ１、２、または３のＣＯＰＹ命令は、実際の位置が記憶される配列の指標を計算するために、それぞれ０、２５６、または５１２に加算されなければならない０～２５５の値を有するバイトがその直

後に設定される。

【００４３】タイプ４のＣＯＰＹ命令は、一連のバイトとしてコード化されたコピー位置を有している。タイプ５のＣＯＰＹ命令は、一連のバイトとしてコード化されたコピー位置と現在位置との差を有している。ＲＥＣＥＮＴキャッシュは、４つの指標を有する配列であり、最新の４つのコピー位置を記憶する。ＣＯＰＹ命令が（コーディング中に）出力されるか、または、（デコーディング中に）処理されたときは必ず、そのコピー位置がキャッシュ内の最も古い位置と入れ替わる。タイプ６（７、８、９）のＣＯＰＹ命令は、キャッシュの指標１（それぞれ、２、３、４）に対応している。そのコピー位置は、対応するキャッシュ指標に記憶されている位置よりも大きいことが保証されており、その差のみがコード化される。

【００４４】タイプ１～９のＡＤＤ命令およびＣＯＰＹ命令の場合、制御バイトの２番目の４ビットが、０でなければ、含まれているデータのサイズをコード化する。これらビットが０であれば、各サイズは、次のバイト列としてコード化される。このようなコーディング方法の結果、ＡＤＤ命令の後に別のＡＤＤ命令が続くことは決してない。ＡＤＤ命令のデータサイズが４以下であり、かつ後に続くＣＯＰＹ命令も小さいことがよくあるが、そのような場合、この２つの命令を単一の制御バイトにマージするうえで、上記の方法は有利である。最初の４ビットである１０～１５の値は、このような結合された命令の組をコード化する。その場合、制御バイトの２番目の４ビットの最初の２ビットにより、ＡＤＤ命令のサイズをコード化し、残りの２ビットによりＣＯＰＹ命令のサイズをコード化する。次に、最初の４ビットの１０～１５の値の一覧を示している。

１０：　ＳＥＬＦとしてコード化されたコピー位置を伴うマージされたＡＤＤ／ＣＯＰＹ命令

１１：　ＨＥＲＥとの差としてコード化されたコピー位置を伴うマージされたＡＤＤ／ＣＯＰＹ命令

１２、１３、１４、１５：　ＲＥＣＥＮＴキャッシュからコード化されたコピー位置を伴うマージされたＡＤＤ／ＣＯＰＹ命令

【００４５】図４は、図２の変形処理情報である４つの命令に関するコード化を示したものである。各命令ごとに、制御バイトの全８ビットが示されている。例えば、２番目の制御バイトの最初の４ビットが０となっているが、これは、バイトによりＡＤＤ命令がコード化されていることを示している。同じバイトの次の４ビットでは、ＡＤＤ命令のサイズが２であることを示している。２つのデータバイト「ｘｙ」が制御バイトの後に置かれている。３つのＣＯＰＹ命令は、すべて、ＳＥＬＦタイプを用いてコード化されており、したがって、そのコピー位置は、制御バイトに続く（示されている値を用いて）バイトによりコード化されている。すべての命令のサイズパラメタは、制御バイト内ですべてコード化できるほど小さい。したがって、この小例は、わずか９バイトを用いるだけで、長さ１７バイトのバージョン２が変形処理情報にコード化できることを示している。

【００４６】<u>一致セグメントの高速計算</u>
図５は、バージョン２をいくつかのセグメントに分割し、ＣＯＰＹ命令およびＡＤＤ命令としてコード化する方法を示している。ただし、このコーディング方法の場合、長さの短い一致は、少なくともそれが一致するデータとしてコード化を行うスペースを取ることから、有効とはいえない。このため、短い一致を無視するように一致方法を調整することができる。ここで用いる一致の最小の長さは４であり、コーディング手続きの本体にある変数ＭＩＮによって示されている。

【００４７】図５の１行目では、探索テーブルＴを初期化して空にする。効率上、Ｔは、衝突をチェーニングしたハッシュテーブルとして保持されている。このデータ構造は標準型であり、「コンピュータアルゴリズムの設計と分析（Ｔｈｅ　Ｄｅｓｉｇｎ　ａｎｄ　Ａｎａｌｙｓｉｓ　ｏｆ　Ｃｏｍｐｕｔｅｒ　Ａｌｇｏｒｉｔｈｍｓ）」（Ａ．Ａｈｏ、Ｊ．Ｈｏｐｃｒｏｆｔ、およびＪ．Ｕｌｌｍａｎ著、１９７４年、Ａｄｄｉｓｏｎ－Ｗｅｓｌｅｙ発行）（１１１～１１２頁）などのデータ構造およびアルゴリズムの教本に説明されている。

【００４８】表Ｔには、２つのバージョンに一定の選択された位置が記載されている。この位置は、手続きｉｎｓｅｒｔ（）によって挿入され、最長の一致データセグメントを高速で探索するために、手続きｓｅａｒｃｈ（）およびｅｘｔｅｎｄ（）で使用される。２行目および３行目では、手続きｐｒｏｃｅｓｓ（）を呼び出し、バージョン１およびバージョン２から位置を選択して表Ｔに挿入する。バージョン２の処理中には、ＣＯＰＹ命令およびＡＤＤ命令の作成も行われる。

【００４９】４～４５行目では、手続きｐｒｏｃｅｓｓ（）を定義する。５行目と６行目では、変数「ｎ」と「ｍ」をバージョン１および処理中のバージョンの長さに初期化する。７行目では、処理中のバージョンの現在位置「ｃ」を０に初期化する。８行目では、ＡＤＤ命令のデータの開始を－１（すなわち、なし）に初期化する。９および１０行目では、位置ｃから始まるデータセグメントの最長の一致に関する位置と長さを初期化する。１１～４２行目では、所定のバージョンを処理するメインループを定義する。１２～１８行目では、ｃから始まるデータセグメントと一致する処理済みの最長データセグメントを計算する。１２行目で呼び出された手続きｓｅａｒｃｈ（）により、長さｌｅｎ＋１の一致を検出する。この手続きでは、位置ｃ＋ｌｅｎ－（ＭＩＮ－１）から始まるＭＩＮバイトを表Ｔの一致している位置を探索するためのキーとして使用する。次に、「ｓｅｑ」のデータと適切なバージョンのデータを逆方向に照

合し、一致がｃからｃ＋ｌｅｎ＋１までのすべてをカバーしているかどうか確認する。

【００５０】このような一致が検出されると、１６行目で呼び出された手続きｅｘｔｅｎｄ（）により、できるだけ長く前方に一致を延長する。１４および１５行目は、一致が全くなく、かつ探索ループから外れる場合に相当する。それ以外の場合は、ループの繰り返しにより、さらに長い一致を検出する。１９および２６行目では、現在の一致していない位置ｃを表Ｔに挿入する。ｓｅｑがバージョン１の場合、挿入される実効値はｃであり、それ以外の場合は、ｃ＋バージョン１の長さ（前述したバージョン２のコーディング位置の規約）である。手続きｉｎｓｅｒｔ（Ｔ，ｓｅｑ，ｐ，原点）では、キーとしてバージョンｓｅｑの位置ｐから始まるＭＩＮバイトを用いて、コード化された位置「ｐ＋原点」を表Ｔに挿入する。

【００５１】２０および２１行目では、未定義であれば、変数ａｄｄをｃに設定し、そこが一致していないデータのセグメントの開始であることを示す。２５行目では、処理ループの次の繰り返しを行うために、現在位置を１だけ前方に移動させる。２７および２８行目は、最長の一致セグメントを検出する事象に相当する。２８および２９行目では、手続きｗｒｉｔｅｉｎｓｔ（）を呼び出して、前述した説明に従って、ＣＯＰＹ命令およびＡＤＤ命令を書き出す。ａｄｄが０または正の値であれば、ｗｒｉｔｅｉｎｓｔ（）への最初の２つの引き数により、ＡＤＤ命令のデータを定義する。第二の２つの引き数では、ＣＯＰＹ命令のパラメタを定義する。この手続きの作用は簡単なので、ここでは説明を省略する。３０～３６行目では、一致したデータセグメントの末尾にあるＭＩＮ－１の位置を表Ｔに挿入する。３７および３８行目では、一致した長さの分だけｃを増加し、ａｄｄを－１にリセットする。４０および４１行目では、処理対象となる充分なデータがない場合に、処理を終了させる。４３および４４行目では、バージョン２から、一致しない最終データをＡＤＤ命令として出力する。

【００５２】図６は、表Ｔに位置を挿入する手続きを示している。図６の２行目は、キーが位置ｐから始まる「ｓｅｑ」列のＭＩＮバイトであることを示している。３行目では、コード化された位置を作成する。４行目では、（ｋｅｙ，ｐｏｓ）の組を表Ｔに挿入する。図７は、一致を探索する手続きを示している。３および４行目では、現在最長となっている一致の長さの末尾のＭＩＮ－１バイトから成る探索キーと一致していない新規の１バイトを作成する。５～１９行目では、可能であれば、一致長さの延長を試行する。この動作は、作成された探索キーと一致する表Ｔの全要素を調べ、キー位置に置かれた現在の一致の一部が検討中の要素の対応する部分と一致しているかどうか確認することによって行われる。このテストは、１７行目で実行される。この結果が

真であれば、１８行目において、ｓｅａｒｃｈ（）により一致の開始位置が返却される。２０行目では、「－１」が返却され、「ｌｅｎ」よりも長い一致がないことを示す。

【００５３】図５の１３行目においてｓｅａｒｃｈ（）への呼び出しが行われた後、一致の長さが、現在、図５の「ｌｅｎ」の値よりも少なくとも１以上長いことが明らかになっている。図８では、ｅｘｔｅｎｄ（）手続きが示されており、この手続きによって、右方向にできるだけ長く一致を延長させる。２～１０行目では、一致を探索する正しい列を設定する。１１～１３行目では、延長を実行する。１４行目では、一致した全体の長さを返却する。図２の例に当てはめると、上記の方法により、同じ図に示されている一連の命令が計算できる。図６では、表Ｔに挿入されるバージョン１およびバージョン２の例の位置が示されている。

【００５４】機密保護の強化

２つのバージョン１およびバージョン２と計算による変換処理情報が与えられた場合、変換処理情報のＡＤＤ命令のみが、バージョン２から生データを取り入れる。このようなデータは、傍受者がバージョン２に関する貴重な情報を得るのに利用可能である。機密保護の必要性が高いアプリケーションでは、情報の漏洩を防ぐために、ＡＤＤ命令を次のように修正することができる。まず初めに、各ＡＤＤ命令を修正して、その位置から始まるデータセグメントが少なくともＡＤＤデータの長さ以上となるようなやり方で、バージョン１から任意に選択される位置となるようなコピーアドレスも持つようにする。そのようにできなければ、ＡＤＤデータはさらに小さい単位に分割することができる。次に、生データは、変換処理情報に出力される前に、このようなデータの選択セグメントからのデータとの排他的論理和がとられる。例えば、図２の同じＡＤＤ命令を用いて、バージョン１の選択位置が２であるとした場合、この位置は、制御バイトの直後に出力され、２データバイト「ｘｙ」は、出力前に、２バイト「ｃｄ」との排他的論理和がとられることになる。

【００５５】デコーディングの場合、出力前に、各データバイトに対して同じ排他的論理和の演算を行わなければならない。排他的論理和の数学的特性により、あるバイトともうひとつの同じバイトとの排他的論理和が２度とられる場合に、元の値が確実に保持されていることから、このような演算が行われる。しかし、これで、傍受者が、安全であるとみなされたバージョン１のコピーをすでに入手していない限り、変換されたデータバイトから何らかの情報を得ることは確実に不可能になる。図７は、このより安全性の高い方法を用いてコード化が行われた図４の命令を示している。これで、ＡＤＤ命令は位置２を有していることになる。データバイト「ｘ」および「ｙ」は、それぞれ、「ｃ」および「ｄ」と排他的論

理和がとられている。本発明の真の精神および範囲を逸脱しない限り、多くの代替および変更を行うことが可能である。特許証による保護を必要とする発明箇所については、特許請求の範囲に定義されている通りである。

【図面の簡単な説明】

【図１】本発明がコンピュータ間のデータ通信にどのように用いられるか示す略図である。

【図１Ａ】類推により本発明によって用いられる原理を示す図のＡである。

【図１Ｂ】類推により本発明によって用いられる原理を示す図のＢである。

【図１Ｃ】類推により本発明によって用いられる原理を示す図のＣである。

【図１Ｄ】類推により本発明によって用いられる原理を示す図のＤである。

【図１Ｅ】類推により本発明によって用いられる原理を示す図のＥである。

【図１Ｆ】類推により本発明によって用いられる原理を示す図のＦである。

【図１Ｇ】類推により本発明によって用いられる原理を示す図のＧである。

【図１Ｈ】類推により本発明によって用いられる原理を示す図のＨである。

【図１Ｉ】類推により本発明によって用いられる原理を示す図のＩである。

【図１Ｊ】本発明の機密保護に関する状況を示す図のＪである。

【図２】データセットの２つのバージョン例とこのバージョンの片方をもう一方のバージョンに変形するための一連の命令を示す図である。

【図２Ａ】図３に示されている手続きの動作を示す図のＡである。

【図２Ｂ】図３に示されている手続きの動作を示す図のＢである。

【図２Ｃ】図３に示されている手続きの動作を示す図のＣである。

【図２Ｄ】図３に示されている手続きの動作を示す図のＤである。

【図２Ｅ】図３に示されている手続きの動作を示す図のＥである。

【図２Ｆ】図５に示されている手続きの動作を示す図のＦである。

【図２Ｇ】図５に示されている手続きの動作を示す図のＧである。

【図２Ｈ】図５に示されている手続きの動作を示す図のＨである。

【図３】擬似Ｃ言語による復号化手続きである。

【図４】図２の命令を実際にコード化したものである。

【図５】コード化の手続きを示す図である。

【図６】図５で使用されたＩＮＳＥＲＴ手続きを示す図である。

【図７】図５で使用されたＳＥＡＲＣＨ手続きを示す図である。

【図８】図５で使用されたＥＸＴＥＮＤ手続きを示す図である。

【図９】図２に例示されたバージョン１およびバージョン２のデータの挿入位置を示す図である。

【図１０】図４のコード化された命令の機密保護が考慮されたバージョンである。

付録

Tue Aug 30 09:21:37 1994

peregrine.zoo.att.com|/n/gryphon/g7/kpv/software/src/lib/vdelta/vdelta.h

```
#ifndef _VDELTA_H
#define _VDELTA_H 1

#ifndef __KPV__
#define __KPV__            1

#ifndef __STD_C
#ifdef __STDC__
#define    __STD_C         1
#else
#if __cplusplus
#define __STD_C            1
#else
#define __STD_C            0
#endif /*__cplusplus*/
#endif /*__STDC__*/
#endif /*__STD_C*/

#ifndef _BEGIN_EXTERNS_
#if __cplusplus
#define _BEGIN_EXTERNS_    extern "C" {
#define _END_EXTERNS_      }
#else
#define _BEGIN_EXTERNS_
#define _END_EXTERNS_
#endif
#endif /*_BEGIN_EXTERNS_*/

#ifndef _ARG_
#if __STD_C
#define _ARG_(x)  x
#else
#define _ARG_(x)  ()
#endif
#endif /*_ARG_*/

#ifndef Void_t
#if __STD_C
#define Void_t             void
#else
#define Void_t             char
#endif
#endif /*Void_t*/

#ifndef NIL
#define NIL(type) ((type)0)
#endif /*NIL*/

#endif /*__KPV__*/

/* user-supplied functions to do io */
typedef struct _vddisc_s  Vddisc_t;
typedef int(*     Vdio_f)_ARG_((int, Void_t*, int, long, Vddisc_t*));
struct _vddisc_s
{ Vdio_f  readf;          /* to read data          */
  Vdio_f  writef;         /* to write data         */
  long    window;         /* window size if any    */
};

```

Tue Aug 30 09:21:37 1994

peregrine.zoo.att.com|/n/gryphon/g7/kpv/software/src/lib/vdelta/vdelta.h

```
/* types that can be given to the IO functions */
#define VD_SOURCE 1       /* io on the source data        */
#define VD_TARGET 2       /* io on the target data        */
#define VD_DELTA  3       /* io on the delta data         */

/* magic header for delta output */
#define VD_MAGIC  "vd01"

_BEGIN_EXTERNS_
extern long        vddelta _ARG_((Void_t*, long, Void_t*, long, Vddisc_t*));
extern long        vdupdate _ARG_((Void_t*, long, Void_t*, long, Vddisc_t*));
_END_EXTERNS_

#endif /*_VDELTA_H*/
```

Wed Aug 10 21:24:44 1994

peregrine.zoo.att.com!/n/gryphon/g7/kpv/Software/src/lib/vdelta/vdelhdr.h　1

```
#ifndef _VDELHDR_H
#define _VDELHDR_H              1

#include  "vdelta.h"

#if __STD_C
#include  <stddef.h>
#else
#include  <sys/types.h>
#endif

#ifdef DEBUG
_BEGIN_EXTERNS_
extern int                      abort();
_END_EXTERNS_
#define ASSERT(p) ((p) ? 0 : abort())
#define DBTOTAL(t,v)            ((t) += (v))
#define DBMAX(m,v)              ((m) = (m) > (v) ? (m) : (v) )
#else
#define ASSERT(p)
#define DBTOTAL(t,v)
#define DBMAX(m,v)
#endif

/* short-hand notations */
#define reg                     register
#define uchar                   unsigned char
#define uint                    unsigned int
#define ulong                   unsigned long

/* default window size - chosen to suit malloc() even on 16-bit machines. */
#define MAXINT                  ((int)(((uint)~0) >> 1))
#define MAXWINDOW ((int)(((uint)~0) >> 2))
#define DFLTWINDOW              (MAXWINDOW <= (1<<14) ? (1<<14) : (1<<16) )
#define HEADER(w) ((w)/4)

#define M_MIN                   4       /* min number of bytes to match */

/* The hash function is s[0]*alpha^3 + s[1]*alpha^2 + s[2]*alpha + s[3] */
#define   ALPHA                 33
#if 0
#define A1(x,t)                 (ALPHA*(x))
#define A2(x,t)                 (ALPHA*ALPHA*(x))
#define A3(x,t)                 (ALPHA*ALPHA*ALPHA*(x))
#else    /* fast multiplication using shifts&adds */
#define A1(x,t)                 ((t = (x)), (t + (t<<5)) )
#define A2(x,t)                 ((t = (x)), (t + (t<<6) + (t<<10)) )
#define A3(x,t)                 ((t = (x)), (t + (t<<5) + ((t+(t<<4))<<6) + ((t+(t<<4))<<11)
)
#endif
#define HINIT(h,s,t)            ((h = A3(s[0],t)), (h += A2(s[1],t)), (h += A1(s[2],t)+s[3]))

#define HNEXT(h,s,t)            ((h -= A3(s[-1],t)), (h = A1(h,t) + s[3]) )

#define EQUAL(s,t)              ((s)[0] == (t)[0] && (s)[1] == (t)[1] && \
                        (s)[2] == (t)[2] && (s)[3] == (t)[3] )

/* Every instruction will start with a control byte.
** For portability, only 8 bits of the byte are used.
```

Wed Aug 10 11:26:44 1994

peregrine.zoo.att.com:/n/gryphon/g7/kpv/Software/src/lib/vdelta/vdelhdr.h

```
** The bits are used as follows:
**          iiii ssss
** ssss: size of data involved.
** iiii: this defines 16 instruction types:
**          0: an ADD instruction.
**          1,2,3: COPY with K_QUICK addressing scheme.
**          4,5: COPY with K_SELF,K_HERE addressing schemes.
**          6,7,8,9: COPY with K_RECENT addressing scheme.
**                  For the above types, ssss if not zero codes the size;
**                  otherwise, the size is coded in subsequent bytes.
**          10,11: merged ADD/COPY with K_SELF,K_HERE addressing
**          12,13,14,15: merged ADD/COPY with K_RECENT addressing.
**                  For merged ADD/COPY instructions, ssss is divided into "cc aa"
**                  where cc codes the size of COPY and aa codes the size of ADD.
*/

#define VD_BITS             8       /* # bits usable in a byte              */

#define S_BITS              4       /* bits for the size field              */
#define I_BITS              4       /* bits for the instruction type        */

/* The below macros compute the coding for a COPY address.
** There are two caches, a "quick" cache of (K_QTYPE*256) addresses
** and a revolving cache of K_RTYPE "recent" addresses.
** First, we look in the quick cache to see if the address is there.
** If so, we use the cache index as the code.
** Otherwise, we compute from 0, the current location and
** the "recent" cache an address that is closest to the being coded address,
** then code the difference. The type is set accordingly.
**
** An invariance is 2*K_MERGE + K_QTYPE + 1 == 16
*/
#define K_RTYPE             4                   /* # of K_RECENT types          */
#define K_QTYPE             3                   /* # of K_QUICK types           */
#define K_MERGE             (K_RTYPE+2)         /* # of types allowing add-copy */
#define K_QSIZE             (K_QTYPE<<VD_BITS)  /* size of K_QUICK cache        */

#define K_QUICK             1                   /* start of K_QUICK types       */
#define K_SELF              (K_QUICK+K_QTYPE)
#define K_HERE              (K_SELF+1)
#define K_RECENT    (K_HERE+1)      /* start of K_RECENT types      */

#define K_DDECL(quick,recent,rhere)         /* cache decls in vdelta        */ \
  int quick[K_QSIZE]; int recent[K_RTYPE]; int rhere/*;*/
#define K_UDECL(quick,recent,rhere)         /* cache decls in vdupdate      */ \
  long quick[K_QSIZE]; long recent[K_RTYPE]; int rhere/*;*/
#define K_INIT(quick,recent,rhere) \
  { quick[rhere=0] = (1<<7); \
    while((rhere += 1) < K_QSIZE) quick[rhere] = rhere + (1<<7); \
    recent[rhere=0] = (1<<8); \
    while((rhere += 1) < K_RTYPE) recent[rhere] = (rhere+1)*(1<<8); \
  }
#define K_UPDATE(quick,recent,rhere,copy) \
  { quick[copy%K_QSIZE] = copy; \
    if((rhere += 1) >= K_RTYPE) rhere = 0; recent[rhere] = copy; \
  }

#define VD_ISCOPY(k)        ((k) > 0 && (k) < (K_RECENT+K_RTYPE) )
#define K_ISMERGE(k)        ((k) >= (K_RECENT+K_RTYPE))
```

Wed Aug 10 21:24:44 1994

peregrine.zoo.att.com:/n/gryphon/g7/kpv/Software/src/lib/vdelta/vdelhdr.h 1

```
#define A_SIZE          ((1<<S_BITS)-1)         /* max local ADD size   */
#define A_ISLOCAL(s)    ((s) <= A_SIZE )        /* can be coded locally */
#define A_LPUT(s) (s)                   /* coded local value    */
#define A_PUT(s)  ((s) - (A_SIZE+1) )   /* coded normal value   */

#define A_ISHERE(i)     ((i) & A_SIZE)          /* locally coded size   */
#define A_LGET(i) ((i) & A_SIZE)
#define A_GET(s)  ((s) + (A_SIZE+1) )

#define C_SIZE          ((1<<S_BITS)+M_MIN-2)   /* max local COPY size  */
#define C_ISLOCAL(s)    ((s) <= C_SIZE )        /* can be coded locally */
#define C_LPUT(s) ((s) - (M_MIN-1) )    /* coded local value    */
#define C_PUT(s)  ((s) - (C_SIZE+1) )   /* coded normal value   */

#define C_ISHERE(i)     ((i) & ((1<<S_BITS)-1)) /* size was coded local */
#define C_LGET(i) (((i) & ((1<<S_BITS)-1)) + (M_MIN-1) )
#define C_GET(s)  ((s) + (C_SIZE+1) )

#define K_PUT(k)  ((k) << S_BITS)
#define K_GET(i)  ((i) >> S_BITS)

/* coding merged ADD/COPY instructions */
#define A_TINY          2               /* bits for tiny ADD            */
#define A_TINYSIZE      (1<<A_TINY)             /* max tiny ADD size    */
#define A_ISTINY(s)     ((s) <= A_TINYSIZE )
#define A_TPUT(s) ((s) - 1)
#define A_TGET(i) (((i) & (A_TINYSIZE-1)) + 1)

#define C_TINY          2               /* bits for tiny COPY           */
#define C_TINYSIZE      ((1<<C_TINY) + M_MIN-1) /* max tiny COPY size   */
#define C_ISTINY(s)     ((s) <= C_TINYSIZE)
#define C_TPUT(s) (((s) - M_MIN) << A_TINY)
#define C_TGET(i) ((((i) >> A_TINY) & ((1<<C_TINY)-1)) + M_MIN )

#define K_TPUT(k) (((k)+K_MERGE) << S_BITS)

#define MEMCPY(to,from,n) \
  switch(n) \
  { default: memcpy((Void_t*)to,(Void_t*)from,(size_t)n); \
             to += n; from += n; break; \
     case 7 : *to++ = *from++; \
     case 6 : *to++ = *from++; \
     case 5 : *to++ = *from++; \
     case 4 : *to++ = *from++; \
     case 3 : *to++ = *from++; \
     case 2 : *to++ = *from++; \
     case 1 : *to++ = *from++; \
     case 0 : break; \
  }

/* Below here is code for a buffered I/O subsystem to speed up I/O */
#define I_SHIFT         7
#define I_MORE          (1<<I_SHIFT)                    /* continuation bit     */
#define I_CODE(n) ((uchar)((n)&(I_MORE-1)) )    /* get lower bits       */

/* structure to do buffered IO */
typedef struct _vdio_s
{ uchar*        next;
  uchar*        endb;
  Vddisc_t*     disc;
```

Wed Aug 10 21:34:44 1994

peregrine.zoo.att.com:/n/gryphon/g7/kpv/Software/src/lib/vdelta/vdelhdr.h

```
	long		here;
	uchar		buf[1024];
} Vdio_t;

#define READF(io)	((io)->disc->readf)
#define WRITEF(io)		((io)->disc->writef)
#define BUF(io)			((io)->buf)
#define BUFSIZE(io)		sizeof((io)->buf)
#define NEXT(io)	((io)->next)
#define ENDB(io)	((io)->endb)
#define DISC(io)	((io)->disc)
#define HERE(io)	((io)->here)
#define RINIT(io,disc)		((io)->endb = ((io)->next = (io)->buf), \
				(io)->here = 0, (io)->disc = (disc) )
#define WINIT(io,disc)		((io)->endb = ((io)->next = (io)->buf) + BUFSIZE(io), \
				(io)->here = 0, (io)->disc = (disc) )
#define REMAIN(io)		(ENDB(io) - NEXT(io))
#define VDPUTC(io,c)		((REMAIN(io) > 0 || (*_Vdflsbuf)(io) > 0) ? \
				(int)(*(io)->next++ = (c)) : -1 )
#define VDGETC(io)		((REMAIN(io) > 0 || (*_Vdfilbuf)(io) > 0) ? \
				(int)(*(io)->next++) : -1 )

typedef struct _vdbufio_s
{ int(*	vdfilbuf)_ARG_((Vdio_t*));
  int(*	vdflsbuf)_ARG_((Vdio_t*));
  ulong(* vdgetu)_ARG_((Vdio_t*, ulong));
  int(*	vdputu)_ARG_((Vdio_t*, ulong));
  int(*	vdread)_ARG_((Vdio_t*, uchar*, int));
  int(*	vdwrite)_ARG_((Vdio_t*, uchar*, int));
} Vdbufio_t;
#define _Vdfilbuf _Vdbufio.vdfilbuf
#define _Vdflsbuf _Vdbufio.vdflsbuf
#define _Vdgetu			_Vdbufio.vdgetu
#define _Vdputu			_Vdbufio.vdputu
#define _Vdread			_Vdbufio.vdread
#define _Vdwrite  _Vdbufio.vdwrite

_BEGIN_EXTERNS_
extern Vdbufio_t  _Vdbufio;
extern Void_t*			memcpy _ARG_((Void_t*, const Void_t*, size_t));
extern Void_t*			malloc _ARG_((size_t));
extern void			free _ARG_((Void_t*));
_END_EXTERNS_

#endif /*_VDELHDR_H*/
```

Wed Sep 28 08:36:11 1994

peregrine.zoo.att.com:/n/gryphon/g7/kpv/Software/src/lib/vdelta/vddelta.c

```
#include	"vdelhdr.h"

/*		Compute a transformation that takes source data to target data
**
**		Written by (Kiem-)Phong Vo, kpv@research.att.com, 5/20/94
*/

#ifdef DEBUG
long		S_copy, S_add;	/* amount of input covered by COPY and ADD	*/
long		N_copy, N_add;	/* # of COPY and ADD instructions		*/
long		M_copy, M_add;	/* max size of a COPY or ADD instruction	*/
long		N_merge;	/* # of merged instructions			*/
#endif

#define MERGABLE(a,c,k) ((a) > 0 && A_ISTINY(a) && \
			(c) > 0 && C_ISTINY(c) && \
			(k) >= K_SELF )

typedef struct _match_s	Match_t;
typedef struct _table_s	Table_t;
struct _match_s
{ Match_t*		next;		/* linked list ptr	*/
};
struct _table_s
{ Vdio_t		io;		/* io structure		*/
  uchar*		src;		/* source string	*/
  int			n_src;
  uchar*		tar;		/* target string	*/
  int			n_tar;
  K_DDECL(quick,recent,rhere);		/* address caches	*/
  Match_t*		base;		/* base of elements	*/
  int			size;		/* size of hash table	*/
  Match_t**		table;		/* hash table		*/
};

/* encode and output delta instructions */
#if __STD_C
static vdputinst(Table_t* tab, uchar* begs, uchar* here, Match_t* match, int n_copy)
#else
static vdputinst(tab, begs, here, match, n_copy)
Table_t*  tab;
uchar*			begs;	/* ADD data if any	*/
uchar*			here;	/* current location	*/
Match_t*  match;	/* best match if any	*/
int			n_copy; /* length of match	*/
#endif
{
	reg int n_add, i_add, i_copy, k_type;
	reg int n, c_addr, copy, best, d;

	n_add = begs ? here-begs : 0;			/* add size		*/
	c_addr = (here-tab->tar)+tab->n_src;		/* current address	*/
	k_type = 0;

	if(match)		/* process the COPY instruction */
	{	/**/DBTOTAL(N_copy,1); DBTOTAL(S_copy,n_copy); DBMAX(M_copy,n_copy);

		best = copy = match - tab->base;
		k_type = K_SELF;
```

Wed Sep 28 08:36:11 1994

peregrine.zoo.att.com:/n/gryphon/g7/kpv/Software/src/lib/vdelta/vddelta.c

```
		if((d = c_addr - copy) < best)
		{	best = d;
			k_type = K_HERE;
		}
		for(n = 0; n < K_RTYPE; ++n)
		{	if((d = copy - tab->recent[n]) < 0 || d >= best)
				continue;
			best = d;
			k_type = K_RECENT+n;
		}
		if(best >= I_MORE && tab->quick[n = copy%K_QSIZE] == copy)
		{	for(d = K_QTYPE-1; d > 0; --d)
				if(n >= (d<<VD_BITS) )
					break;
			best = n - (d<<VD_BITS); /**/ASSERT(best < (1<<VD_BITS));
			k_type = K_QUICK+d;
		}

		/**/ASSERT(best >= 0);
		/**/ASSERT((k_type+K_MERGE) < (1<<I_BITS) );

		/* update address caches */
		K_UPDATE(tab->quick,tab->recent,tab->rhere,copy);

		/* see if mergable to last ADD instruction */.
		if(MERGABLE(n_add,n_copy,k_type) )
		{	/**/DBTOTAL(N_merge,1);
			i_add = K_TPUT(k_type)|A_TPUT(n_add)|C_TPUT(n_copy);
		}
		else
		{	i_copy = K_PUT(k_type);
			if(C_ISLOCAL(n_copy) )
				i_copy |= C_LPUT(n_copy);
		}
	}

	if(n_add > 0)
	{	/**/DBTOTAL(N_add,1); DBTOTAL(S_add,n_add); DBMAX(M_add,n_add);

		if(!MERGABLE(n_add,n_copy,k_type) )
			i_add = A_ISLOCAL(n_add) ? A_LPUT(n_add) : 0;

		if(VDPUTC((Vdio_t*)tab,i_add) < 0 )
			return -1;
		if(!A_ISLOCAL(n_add) &&
		   (*_Vdputu)((Vdio_t*)tab, (ulong)A_PUT(n_add)) < 0 )
			return -1;
		if((*_Vdwrite)((Vdio_t*)tab, begs, n_add) < 0 )
			return -1;
	}

	if(n_copy > 0)
	{	if(!MERGABLE(n_add,n_copy,k_type) && VDPUTC((Vdio_t*)tab,i_copy) < 0
)
			return -1;

		if(!C_ISLOCAL(n_copy) &&
		   (*_Vdputu)((Vdio_t*)tab, (ulong)C_PUT(n_copy)) < 0 )
			return -1;

```

Wed Sep 28 08:36:11 1994

peregrine.zoo.att.com!/n/gryphon/g7/kpv/SoftWare/src/lib/vdelta/vddelta.c

```
			if(k_type >= K_QUICK && k_type < (K_QUICK+K_QTYPE) )
			{	if(VDPUTC((Vdio_t*)tab,(uchar)best) < 0 )
					return -1;
			}
			else
			{	if((*_Vdputu)((Vdio_t*)tab, (ulong)best) < 0 )
					return -1;
			}
	}
	else
	{	if((*_Vdflsbuf)((Vdio_t*)tab) < 0)
			return -1;
	}

	return 0;
}


/* Fold a string */
#if __STD_C
static vdfold(Table_t* tab, int output)
#else
static vdfold(tab, output)
Table_t*	tab;
int		output;
#endif
{
	reg ulong	key, n;
	reg uchar	*s, *sm, *ends, *ss, *heads;
	reg Match_t	*m, *list, *curm, *bestm;
	reg uchar	*add, *endfold;
	reg int		head, len, n_src = tab->n_src;
	reg int		size = tab->size;
	reg uchar	*src = tab->src, *tar = tab->tar;
	reg Match_t	*base = tab->base, **table = tab->table;

	if(!output)
	{	if(tab->n_src < M_MIN)
			return 0;
		endfold = (s = src) + tab->n_src;
		curm = base;
	}
	else
	{	endfold = (s = tar) + tab->n_tar;
		curm = base+n_src;
		if(tab->n_tar < M_MIN)
			return vdputinst(tab,s,endfold,NIL(Match_t*),0);
	}

	add = NIL(uchar*);
	bestm = NIL(Match_t*);
	len = M_MIN-1;
	HINIT(key,s,n);
	for(;;)
	{	for(;;) /* search for the longest match */
		{	if(!(m = table[key&size]) )
				goto endsearch;
			list = m = m->next;	/* head of list */

			if(bestm) /* skip over past elements */
```

Wed Sep 28 08:36:11 1994

peregrine.zoo.att.com:/n/gryphon/g7/kpv/Software/src/lib/vdelta/vddelta.c

```
			{	for(;;)
				{	if(m >= bestm+len)
						break;
					if((m = m->next) == list)
						goto endsearch;
				}
			}

			head = len - (M_MIN-1); /* header before the match */
			heade = s+head;
			for(;;)
			{	if((n = m-base) < n_src)
				{	if(n < head)
						goto next;
					sm = src + n;
				}
				else
				{	if((n -= n_src) < head)
						goto next;
					sm = tar + n;
				}

				/* make sure that the M_MIN bytes match */
				if(!EQUAL(heade,sm))
					goto next;

				/* make sure this is a real match */
				for(sm -= head, ss = s; ss < heade; )
					if(*sm++ != *ss++)
						goto next;
				ss += M_MIN;
				sm += M_MIN;
				ends = endfold;
				if((m-base) < n_src && (n = (src+n_src)-sm) < (ends-s
))
					ends = s+n;
				for(; ss < ends; ++ss, ++sm)
					if(*sm != *ss)
						goto extend;
				goto extend;

			next:	if((m = m->next) == list )
					goto endsearch;
			}

		extend: bestm = m-head;
			n = len;
			len = ss-s;
			if(ss >= endfold)	/* already match everything */
				goto endsearch;

			/* check for a longer match */
			ss -= M_MIN-1;
			if(len == n+1)
				HNEXT(key,ss,n);
			else	HINIT(key,ss,n);
		}

	endsearch:
		if(bestm)
```

Wed Sep 28 08:16:11 1994

peregrine.zoo.att.com!/n/gryphon/g7/kpv/Software/src/lib/vdelta/vddelta.c

```
	{	if(output && vdputinst(tab,add,s,bestm,len) < 0)
				return -1;

			/* add a sufficient number of suffices */
			ends = (s += len);
			ss = ends - (M_MIN-1);
			if(!output)
				curm = base - (ss-src);
			else	curm = base + n_src + (ss-tar);

			len = M_MIN-1;
			add = NIL(uchar*);
			bestm = NIL(Match_t*);
		}
		else
		{	if(!add)
				add = s;
			ss = s;
			ends = (s += 1);	/* add one prefix */
		}

		if(ends > (endfold - (M_MIN-1)) )
			ends = endfold - (M_MIN-1);

		if(ss < ends) for(;;)	/* add prefices/suffices */
		{	n = keysize;
			if(!(m = table[n]) )
				curm->next = curm;
			else
			{	curm->next = m->next;
				m->next = curm;
			}
			table[n] = curm++;

			if((ss += 1) >= ends)
				break;
			HNEXT(key,ss,n);
		}

		if(s > endfold-M_MIN)	/* too short to match */
		{	if(!add)
				add = s;
			break;
		}

		HNEXT(key,s,n);
	}

	if(output)	/* flush output */
		return vdputinst(tab,add,endfold,NIL(Match_t*),0);
	return 0;
}


#if __STD_C
long vddelta(Void_t* src, long n_src, Void_t* tar, long n_tar, Vddisc_t* disc)
#else
long vddelta(src, n_src, tar, n_tar, disc)
Void_t*		src;	/* source string if not NULL	*/
long		n_src;	/* length of source data	*/
```

Wed Sep 28 08:36:11 1994

| paragrine.zoo.att.com!/n/gryphon/g7/kpv/Software/src/lib/vdelta/vddelta.c |

```
Void_t*          tar;    /* target string if not NULL     */
long             n_tar;  /* length of target data         */
Vddisc_t* disc;  /* IO discipline                  */
#endif
{
  reg int          size, k, n, window;
  reg long         p;
  Table_t          tab;

  if(!disc || !disc->writef ||
     (!disc->readf && ((n_tar > 0 && !tar) || (n_src > 0 && !src) ) ) )
          return -1;

  if(n_tar < 0)
          return -1;
  if(n_src < 0)
          n_src = 0;

  tab.n_src = tab.n_tar = tab.size = 0;
  tab.tar = tab.src = NIL(uchar*);
  tab.base = NIL(Match_t*);
  tab.table = NIL(Match_t**);
  WINIT(&tab.io,disc);

  if(disc->window <= 0)
          window = DFLTWINDOW;
  else if(disc->window > MAXWINDOW)
          window = MAXWINDOW;
  else    window = (int)disc->window;
  if((long)window > n_tar)
          window = (int)n_tar;
  if(n_src > 0 && (long)window > n_src)
          window = (int)n_src;

  /* try to allocate working space */
  while(window > 0)
  {       /* space for the target string */
          size = (n_tar == 0 || tar) ? 0 : window;
          if((long)size > n_tar)
                  size = (int)n_tar;
          if(size > 0 && !(tab.tar = (uchar*)malloc(size*sizeof(uchar))) )
                  goto reduce_window;

          /* space for sliding header or source string */
          if(n_src <= 0)  /* compression only */
                  size = tar ? 0 : HEADER(window);
          else            /* differencing */
          {       size = src ? 0 : window;
                  if((long)size > n_src)
                          size = (int)n_src;
          }
          if(size > 0 && !(tab.src = (uchar*)malloc(size*sizeof(uchar))) )
                  goto reduce_window;

          /* space for the hash table elements */
          size = window < n_tar ? window : (int)n_tar;
          if(n_src <= 0)
                  size += window < n_tar ? HEADER(window) : 0;
          else    size += window < n_src ? window : (int)n_src;
          if(!(tab.base = (Match_t*)malloc(size*sizeof(Match_t))) )
```

Wed Sep 28 08:36:11 1994

peregrine.research.att.com:/n/gryphon/g7/kpv/Software/src/lib/vdelta/vddelta.c

```
				goto reduce_window;

		/* space for the hash table */
		n = size/2;
		do (size = n); while((n &= n-1) != 0);
		if(size < 64)
			size = 64;
		while(!(tab.table = (Match_t**)malloc(size*sizeof(Match_t*))) )
			if((size >>= 1) <= 0)
				goto reduce_window;

		/* if get here, successful */
		tab.size = size-1;
		break;

	reduce_window:
		if(tab.tar)
		{	free((Void_t*)tab.tar);
			tab.tar = NIL(uchar*);
		}
		if(tab.src)
		{	free((Void_t*)tab.src);
			tab.src = NIL(uchar*);
		}
		if(tab.base)
		{	free((Void_t*)tab.base);
			tab.base = NIL(Match_t*);
		}
		if((window >>= 1) <= 0)
			return -1;
	}

	/* amount processed */
	n = 0;

	/* output magic bytes and sizes */
	for(k = 0; VD_MAGIC[k]; k++)
		;
	if((*_Vdwrite)(&tab.io,(uchar*)VD_MAGIC,k) != k ||
	   (*_Vdputu)(&tab.io,(ulong)n_tar) <= 0 ||
	   (*_Vdputu)(&tab.io,(ulong)n_src) <= 0 ||
	   (*_Vdputu)(&tab.io,(ulong)window) <= 0 )
		goto done;

	/* do one window at a time */
	while(n < n_tar)
	{	/* prepare the source string */
		if(n_src <= 0)	/* data compression */
		{	if(n <= 0)
				tab.n_src = 0;
			else
			{	size = HEADER(window);
				if(tar)
					tab.src = tab.tar + tab.n_tar - size;
				else	memcpy((Void_t*)tab.src,
						(Void_t*)(tab.tar + tab.n_tar - size),
						size );
				tab.n_src = size;
			}
		}
```

Wed Sep 28 08:36:11 1994

peregrine.zoo-att.com!/n/gryphon/g7/kpv/Software/src/lib/vdelta/vddelta.c

```
	else	/* data differencing */
	{	if(n < n_src)
		{	if(n+window > n_src)
				p = n_src-window;
			else	p = n;
			if(src)
				tab.src = (uchar*)src + p;
			else
			{	size = (*disc->readf)(VD_SOURCE, tab.src,
							window, p, disc);
				if(size != window)
					goto done;
			}
		} /* else use last window */

		tab.n_src = window;
	}

	/* prepare the target string */
	size = (n_tar-n) < window ? (int)(n_tar-n) : window;
	tab.n_tar = size;
	if(tar)
		tab.tar = (uchar*)tar + n;
	else
	{	size = (*disc->readf)(VD_TARGET, tab.tar, size, (long)n, disc
);
		if((long)size != tab.n_tar)
			goto done;
	}

	/* reinitialize table before processing */
	for(k = tab.size; k >= 0; --k)
		tab.table[k] = NIL(Match_t*);
	K_INIT(tab.quick,tab.recent,tab.rhere);

	if(tab.n_src > 0 && vdfold(&tab,0) < 0)
		goto done;
	if(vdfold(&tab,1) < 0)
		goto done;

	n += size;
  }

done:
  if(!tar && tab.tar)
	free((Void_t*)tab.tar);
  if(tab.src && ((n_src <= 0 && !tar) || (n_src > 0 && !src) ) )
	free((Void_t*)tab.src);
  if(tab.base)
	free((Void_t*)tab.base);
  if(tab.table)
	free((Void_t*)tab.table);

  return n;
}
```

Sun Aug 14 11.53:45 1994

peregrine.zoo.att.com:/n/gryphon/g7/kpv/Software/src/lib/vdelta/vdupdate.c

```
#include	"vdelhdr.h"


/*	Apply the transformation source->target to reconstruct target
**	This code is designed to work even if the local machine has
**	word size smaller than that of the machine where the delta
**	was computed. A requirement is that "long" on the local
**	machine must be large enough to hold source and target sizes.
**	It is also assumed that if an array is given, the size of
**	that array in bytes must be storable in an "int". This is
**	used in various cast from "long" to "int".
**
**	Written by (Kiem-)Phong Vo, kpv@research.att.com, 5/20/94
*/
typedef struct _table_s
{ Vdio_t		io;		/* io structure			*/
  uchar*		src;		/* source string		*/
  long		n_src;
  uchar*		tar;		/* target string		*/
  long		n_tar;
  long		s_org;		/* start of window in source	*/
  long		t_org;		/* start of window in target	*/
  uchar		data[128];	/* buffer for data transferring */
  char		s_alloc;	/* 1 if source was allocated	*/
  char		t_alloc;	/* 1 if target was allocated	*/
  char		compress;	/* 1 if compressing only	*/
  K_UDECL(quick,recent,rhere);	/* address caches		*/
} Table_t;

#if __STD_C
static vdunfold(Table_t* tab)
#else
static vdunfold(tab)
Table_t*	tab;
#endif
{
  reg long	size, copy;
  reg int	inst, k_type, n, r;
  reg uchar	*tar, *src, *to, *fr;
  reg long	t, c_addr, n_tar, n_src;
  reg Vdio_f	readf, writef;
  reg Vddisc_t*	disc;

  n_tar = tab->n_tar;
  tar = tab->tar;
  n_src = tab->n_src;
  src = tab->src;

  disc = tab->io.disc;
  readf = disc->readf;
  writef = disc->writef;

  for(t = 0, c_addr = n_src; t < n_tar; )
  {	if((inst = VDGETC((Vdio_t*)tab)) < 0)
		return -1;
	k_type = K_GET(inst);

	if(!VD_ISCOPY(k_type))
	{	if(K_ISMERGE(k_type))	/* merge/add instruction	*/
```

Sun Aug 14 11:55:45 1994

| peregrine.zoo.att.com:/n/gryphon/s7/kpv/software/sxo/lib/vdelta/vdupdate.c |

```
                size = A_TGET(inst);
        else if(A_ISHERE(inst)) /* locally coded ADD size        */
                size = A_LGET(inst);
        else                    /* non-local ADD size            */
        {       if((size = VDGETC((Vdio_t*)tab)) < 0)
                        return -1;
                if(size >= I_MORE &&
                   (size = (long)(*_Vdgetu)((Vdio_t*)tab,size)) < 0)
                        return -1;
                size = A_GET(size);
        }
        if((t+size) > n_tar)    /* out of sync  */
                return -1;
        c_addr += size;

        /* copy data from the delta stream to target */
        for(;;)
        {       if(!tar)
                {       if((long)(n = sizeof(tab->data)) > size)
                                n = (int)size;
                        if((*_Vdread)((Vdio_t*)tab,tab->data,n) != n
)
                                return -1;
                        r = (*writef)(VD_TARGET,
                                (Void_t*)tab->data, n,
                                tab->t_org+t, disc);
                        if(r != n)
                                return -1;
                }
                else
                {       n = (int)size;
                        if((*_Vdread)((Vdio_t*)tab,tar+t,n) != n)
                                return -1;
                }
                t += n;
                if((size -= n) <= 0)
                        break;
        }

        if(K_ISMERGE(k_type))
        {       size = C_TGET(inst);
                k_type -= K_MERGE;
                goto do_copy;
        }
}
else
{       if(C_ISHERE(inst))      /* locally coded COPY size */
                size = C_LGET(inst);
        else
        {       if((size = VDGETC((Vdio_t*)tab)) < 0)
                        return -1;
                if(size >= I_MORE &&
                   (size = (long)(*_Vdgetu)((Vdio_t*)tab,size)) < 0)
                        return -1;
                size = C_GET(size);
        }
do_copy:
        if((t+size) > n_tar)    /* out of sync */
                return -1;
```

Sun Aug 14 11:55:45 1994

| peregrine.zoo.att.com:/n/gryphon/g7/kpv/Software/src/lib/vdelta/vdupdate.c |

```
		if((copy = VDGETC((Vdio_t*)tab)) < 0)
			return -1;
		if(k_type >= K_QUICK && k_type < (K_QUICK+K_QTYPE) )
			copy = tab->quick[copy + ((k_type-K_QUICK)<<VD_BITS)];

		else
		{	if(copy >= I_MORE &&
			   (copy = (long)(*_Vdgetu)((Vdio_t*)tab,copy)) < 0)
				return -1;
			if(k_type >= K_RECENT && k_type < (K_RECENT+K_RTYPE))

				copy += tab->recent[k_type - K_RECENT];
			else if(k_type == K_HERE)
				copy = c_addr - copy;
			/* else k_type == K_SELF */
		}
		K_UPDATE(tab->quick,tab->recent,tab->rhere,copy);
		c_addr += size;

		if(copy < n_src)	/* copy from source data */
		{	if((copy+size) > n_src)	/* out of sync */
				return -1;
			if(src)
			{	n = (int)size;
				fr = src+copy;
				if(tar)
				{	to = tar+t;
					MEMCPY(to,fr,n);
				}
				else
				{	r = (*writef)(VD_TARGET, (Void_t*)fr, n,

						tab->t_org+t, disc);
					if(r != n)
						return -1;
				}
				t += n;
			}
			else
			{	if(tab->compress)
				{	copy += tab->t_org - tab->n_src;
					inst = VD_TARGET;
				}
				else
				{	copy += tab->s_org;
					inst = VD_SOURCE;
				}
				for(;;)
				{	if(tar)
					{	n = (int)size;
						r = (*readf)(inst,
							(Void_t*)(tar+t), n,
							copy, disc);
					}
					else
					{	n = sizeof(tab->data);
						if((long)n > size)
							n = (int)size;
						r = (*readf)(inst,
							(Void_t*)tab->data,n,
```

Sun Aug 14 11:55:45 1994

peregrine.zoo.att.com!/n/gryphon/g7/kpv/Software/src/lib/vdelta/vdupdate.c

```
                                                copy, disc);
                                        if(r != n)
                                                return -1;
                                        r = (*writef)(VD_TARGET,
                                                (Void_t*)tab->data, n
,
                                                tab->t_org-t, disc);
                                }
                                if(r != n)
                                        return -1;
                                t -= n;
                                if((size -= n) <= 0)
                                        break;
                                copy += n;
                        }
                }
        }
        else    /* copy from target data */
        {       copy -= n_src;
                if(copy >= t || (copy+size) > n_tar) /* out-of-sync *
/
                        return -1;
                for(;;) /* allow for copying overlapped data */
                {       reg long        s, a;
                        if((s = t-copy) > size)
                                s = size;
                        if(tar)
                        {       to = tar+t; fr = tar+copy; n = (int)s
;
                                MEMCPY(to,fr,n);
                                t += n;
                                goto next;
                        }

                        /* hard read/write */
                        a = copy;
                        for(;;)
                        {       if((long)(n = sizeof(tab->data)) > s)
                                        n = (int)s;
                                r = ( *readf)(VD_TARGET,
                                                (Void_t*)tab->data, n,
                                                a + tab->t_org, disc );
                                if(r != n)
                                        return -1;
                                r = (*writef)(VD_TARGET,
                                                (Void_t*)tab->data, n,
                                                t + tab->t_org, disc );
                                if(r != n)
                                        return -1;
                                t += n;
                                if((s -= n) <= 0)
                                        break;
                                a += n;
                        }

                next:   if((size -= s) == 0)
                                break;
                }
        }
}
```

Sun Aug 14 11:55:45 1994

| peregrine.spo.att.com!/n/gryphon/g7/kpv/Software/src/lib/vdelta/vdupdate.c |

```
  }

  return 0;
}

#if __STD_C
long vdupdate(Void_t* src, long n_src, Void_t* tar, long n_tar, Vddisc_t* disc)
#else
long vdupdate(src, n_src, tar, n_tar, disc)
Void_t*         src;    /* source string if any */
long            n_src;  /* length of src        */
Void_t*         tar;    /* target space if any  */
long            n_tar;  /* size of tar          */
Vddisc_t* disc;
#endif
{
  Table_t tab;
  uchar   *data, magic[8];
  int     n, r;
  long    t, p, window;
  Vdio_f  readf, writef;

  if(!disc || !disc->readf || (!tar && !disc->writef) )
          return -1;
  readf = disc->readf;
  writef = disc->writef;

  /* initialize I/O buffer */
  RINIT(&tab.io,disc);

  /* check magic header */
  data = (uchar*)(VD_MAGIC);
  for(n = 0; data[n]; ++n)
          ;
  if((*_Vdread)(&tab.io,magic,n) != n)
          return -1;
  for(n -= 1; n >= 0; --n)
          if(data[n] != magic[n])
                  return -1;

  /* get true target size */
  if((t = (long)(*_Vdgetu)(&tab.io,0)) < 0 || (tar && n_tar != t) )
          return -1;
  n_tar = t;

  /* get true source size */
  if((t = (long)(*_Vdgetu)(&tab.io,0)) < 0 || (src && n_src != t) )
          return -1;
  n_src = t;

  /* get window size */
  if((window = (long)(*_Vdgetu)(&tab.io,0)) < 0)
          return -1;

  tab.compress = n_src == 0 ? 1 : 0;

  /* if we have space, it'll be faster to unfold */
  tab.tar = tab.src = NIL(uchar*);
  tab.t_alloc = tab.s_alloc = 0;

```

Sun Aug 14 11:55:45 1994

peregrine.zoo.att.com:/n/gryphon/g7/kpv/Software/src/lib/vdelta/vdupdate.c

```
if(n_tar > 0 && !tar && window < (long)MAXINT)
        n = (int)window;
else    n = 0;
if(n > 0 && (tab.tar = (uchar*)malloc(n*sizeof(uchar))) )
        tab.t_alloc = 1;

if(n_src > 0 && !src && window < (long)MAXINT)
        n = (int)window;
else if(n_src == 0 && window < n_tar && !tar && HEADER(window) < (long)MAXINT
)
        n = (int)HEADER(window);
else    n = 0;
if(n > 0 && (tab.src = (uchar*)malloc(n*sizeof(uchar))) )
        tab.s_alloc = 1;

for(t = 0; t < n_tar; )
{       tab.t_org = t;  /* current location in target stream */

        if(n_src == 0)  /* data compression */
        {       tab.s_org = 0;

                if(t == 0)
                        tab.n_src = 0;
                else
                {       tab.n_src = HEADER(window);
                        p = t - tab.n_src;
                        if(tar)
                                tab.src = (uchar*)tar + p;
                        else if(tab.src)
                        {       n = (int)tab.n_src;
                                if(tab.tar)
                                {       data = tab.tar + tab.n_tar - n;
                                        memcpy((Void_t*)tab.src,(Void_t*)data
,n);

                                }
                                else
                                {       r = (*readf)(VD_TARGET,tab.src,n,p,di
sc);
                                        if(r != n)
                                                goto done;
                                }
                        }
                }
        }
        else    /* data differencing */
        {       tab.n_src = window;
                if(t < n_src)
                {       if((t+window) > n_src)
                                p = n_src-window;
                        else    p = t;

                        tab.s_org = p;

                        if(src)
                                tab.src = (uchar*)src + p;
                        else if(tab.src)
                        {       n = (int)tab.n_src;
                                r = (*readf)(VD_SOURCE,tab.src,n,p,disc);
                                if(r != n)
                                        goto done;
```

Sun Aug 14 11:55:45 1994

peregrine.zoo.att.com:/n/gryphon/g7/kpv/Software/src/lib/vdelta/vdupdate.c

```
                        }
                }
        }

        if(tar)
                tab.tar = (uchar*)tar+t;
        tab.n_tar = window < (n_tar-t) ? window : (n_tar-t);

        K_INIT(tab.quick,tab.recent,tab.rhere);
        if(vdunfold(&tab) < 0)
                goto done;
        if(!tar && tab.tar)
        {       p = (*writef)(VD_TARGET,(Void_t*)tab.tar,(int)tab.n_tar,t,dis
c);
                if(p != tab.n_tar)
                        goto done;
        }

        t += tab.n_tar;
}

done:
  if(tab.t_alloc)
        free((Void_t*)tab.tar);
  if(tab.s_alloc)
        free((Void_t*)tab.src);

  return t;
}
```

Sun Aug 14 11:56:34 1994

| peregrine.zoo.att.com:/n/gryphon/g7/kpv/Software/src/lib/vdelta/vdio.c |

```
#include "vdelhdr.h"


#if __STD_C
static _vdfilbuf(reg Vdio_t* io)
#else
static _vdfilbuf(io)
reg Vdio_t*	io;
#endif
{ reg int n;

  if((n = (*READF(io))(VD_DELTA,BUF(io),BUFSIZE(io),HERE(io),DISC(io))) > 0)
  {	ENDB(io) = (NEXT(io) = BUF(io)) + n;
	HERE(io) += n;
  }
  return n;
}

#if __STD_C
static _vdflsbuf(reg Vdio_t* io)
#else
static _vdflsbuf(io)
reg Vdio_t*	io;
#endif
{ reg int n;

  if((n = NEXT(io) - BUF(io)) > 0 &&
     (*WRITEF(io))(VD_DELTA,BUF(io),n,HERE(io),DISC(io)) != n)
	return -1;

  HERE(io) += n;
  NEXT(io) = BUF(io);
  return BUFSIZE(io);
}

#if __STD_C
static ulong _vdgetu(reg Vdio_t* io, reg ulong v)
#else
static ulong _vdgetu(io,v)
reg Vdio_t*	io;
reg ulong v;
#endif
{ reg int	c;

  for(v &= I_MORE-1;;)
  {	if((c = VDGETC(io)) < 0)
		return (ulong)(-1L);
	if(!(c&I_MORE) )
		return ((v<<I_SHIFT) | c);
	v = (v<<I_SHIFT) | (c & (I_MORE-1));
  }
}

#if __STD_C
static _vdputu(reg Vdio_t* io, ulong v)
#else
static _vdputu(io, v)
reg Vdio_t*	io;
reg ulong v;
```

Sun Aug 14 11:54:34 1994

_peregrine.zoo.att.com:/n/gryphon/g7/kpv/Software/src/lib/vdelta/vdio.c

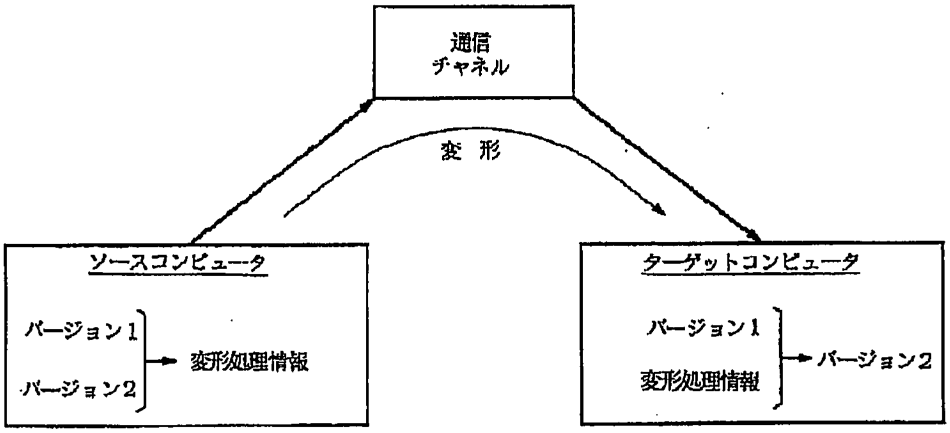
```
#endif
{
  reg uchar       *s, *next;
  reg int         len;
  uchar           c[sizeof(ulong)+1];

  s = next = &c[sizeof(c)-1];
  *s = I_CODE(v);
  while((v >>= I_SHIFT) )
          *--s = I_CODE(v)|I_MORE;
  len = (next-s) + 1;

  if(REMAIN(io) < len && _vdflsbuf(io) <= 0)
          return -1;

  next = io->next;
  switch(len)
  {
  default: memcpy((Void_t*)next,(Void_t*)s,len); next += len; break;
  case 3: *next++ = *s++;
  case 2: *next++ = *s++;
  case 1: *next++ = *s;
  }
  io->next = next;

  return len;
}

#if __STD_C
static _vdread(Vdio_t* io, reg uchar* s, reg int n)
#else
static _vdread(io, s, n)
Vdio_t*         io;
reg uchar*      s;
reg int         n;
#endif
{
  reg uchar*      next;
  reg int         r, m;

  for(m = n; m > 0; )
  {       if((r = REMAIN(io)) <= 0 && (r = _vdfilbuf(io)) <= 0)
                  break;
          if(r > m)
                  r = m;

          next = io->next;
          MEMCPY(s,next,r);
          io->next = next;

          m -= r;
  }
  return n-m;
}

#if __STD_C
static _vdwrite(Vdio_t* io, reg uchar* s, reg int n)
#else
static _vdwrite(io, s, n)
Vdio_t*         io;
```

Sun Aug 14 11:54:34 1994

_peregrine.zoo.att.com:/n/gryphon/g7/kpv/Software/src/lib/vdelta/vdio.c

```
reg uchar*      s;
reg int         n;
#endif
{
  reg uchar*      next;
  reg int         w, m;

  for(m = n; m > 0; )
  {       if((w = REMAIN(io)) <= 0 && (w = _vdflsbuf(io)) <= 0)
                  break;
          if(w > m)
                  w = m;

          next = io->next;
          MEMCPY(next,s,w);
          io->next = next;

          m -= w;
  }
  return n-m;
}


Vdbufio_t _Vdbufio =
{ _vdfilbuf,
  _vdflsbuf,
  _vdgetu,
  _vdputu,
  _vdread,
  _vdwrite
};
```

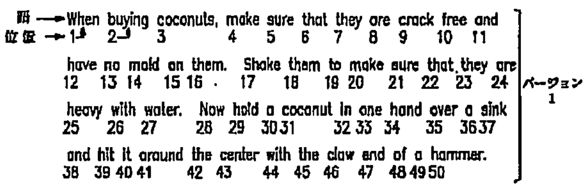
【図1】



【図4】

01000100	0
00000010	x y
01000110	20
01000101	9

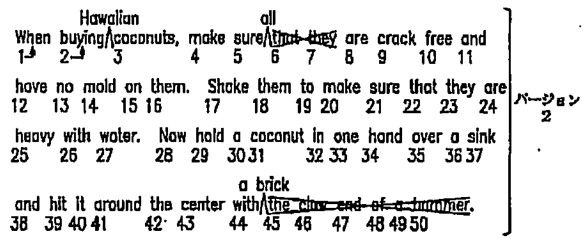
【図1A】



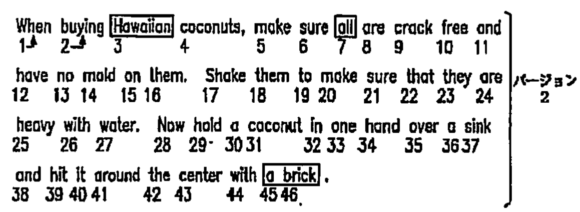
単語表

1 When	16 them	31 it
2 buying	17 Shake	32 around
3 coconuts	18 to	33 the
4 make	19 heavy	34 center
5 sure	20 with	35 claw
6 that	21 water	36 end
7 they	22 Now	37 of
8 are	23 hold	38 hammer
9 crack	24 a	
10 free	25 in	
11 and	26 one	
12 have	27 hand	
13 no	28 over	
14 mold	29 sink	
15 on	30 hit	

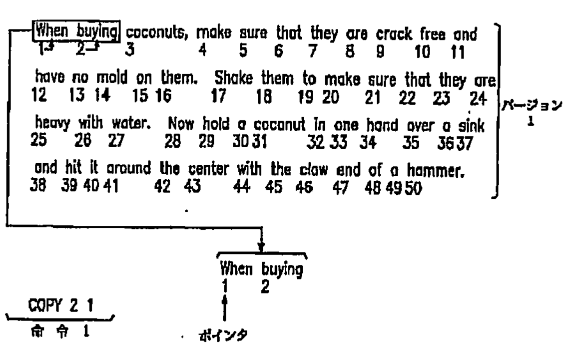
【図1B】



【図1C】



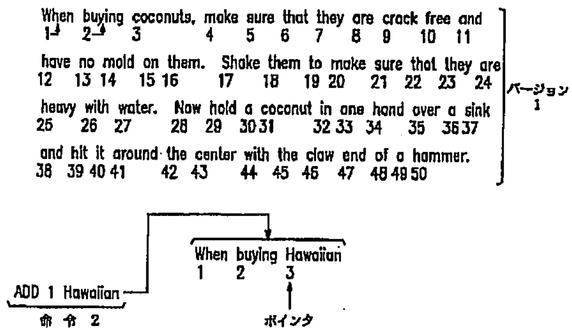
【図1D】



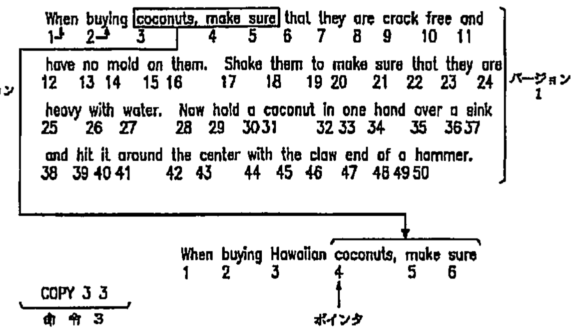
単語表

1 When	16 them	31 it
2 buying	17 Shake	32 around
3 coconuts	18 to	33 the
4 make	19 heavy	34 center
5 sure	20 with	35 claw
6 that	21 water	36 end
7 they	22 Now	37 of
8 are	23 hold	38 hammer
9 crack	24 a	39 Hawaiian 新語
10 free	25 in	40 all 新語
11 and	26 one	41 brick 新語
12 have	27 hand	
13 no	28 over	
14 mold	29 sink	
15 on	30 hit	

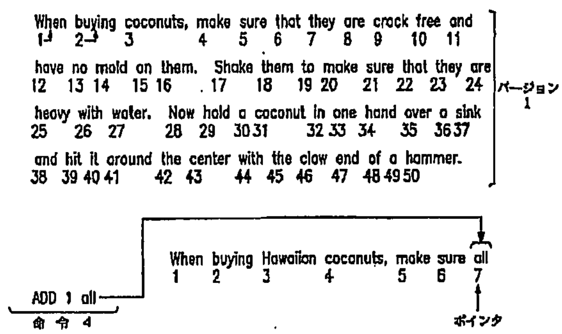
【図1E】



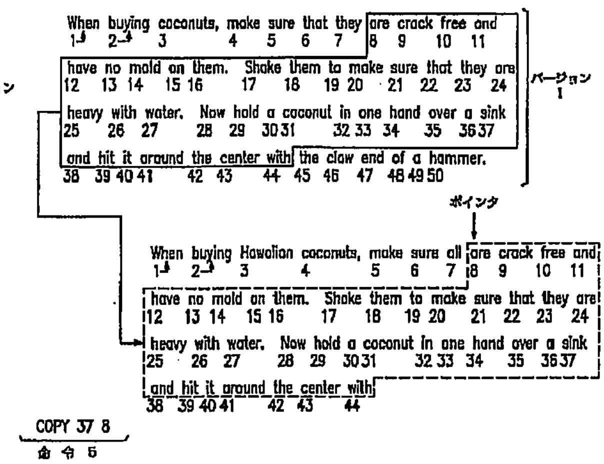
【図1F】



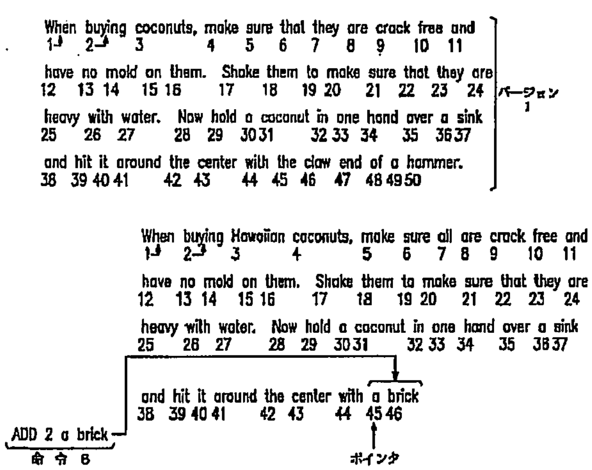
【図1G】



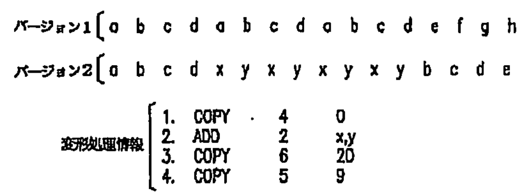
【図1H】



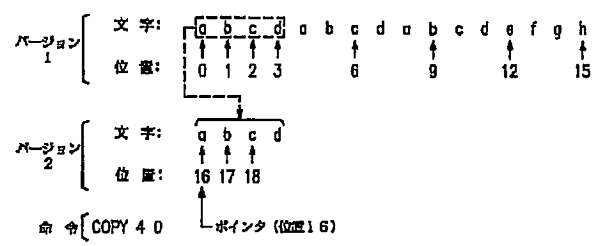
【図1I】



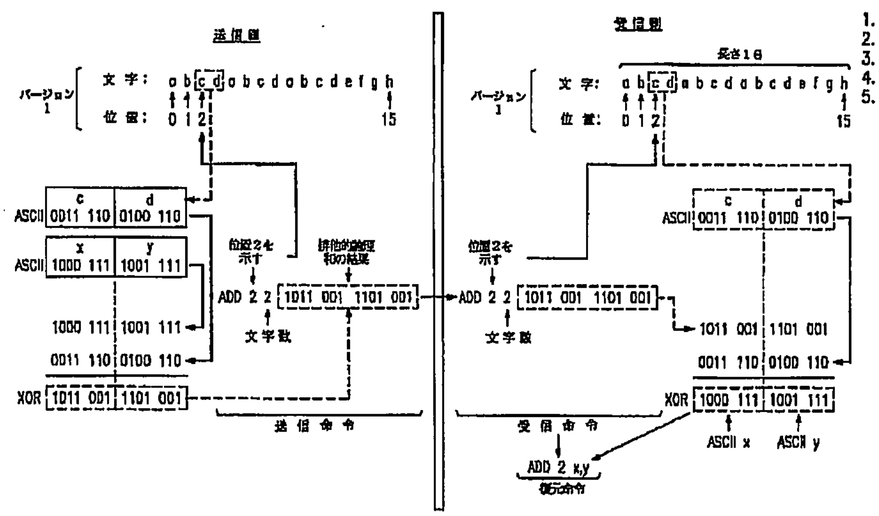
【図2】



【図2B】



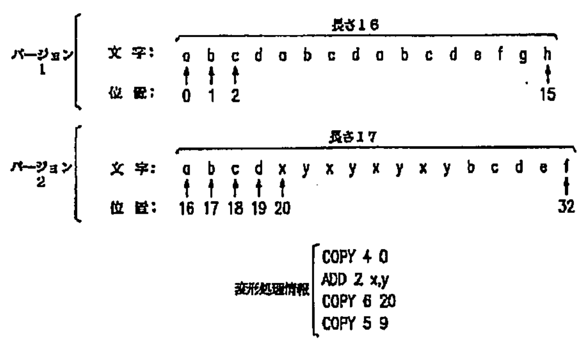
【図1J】



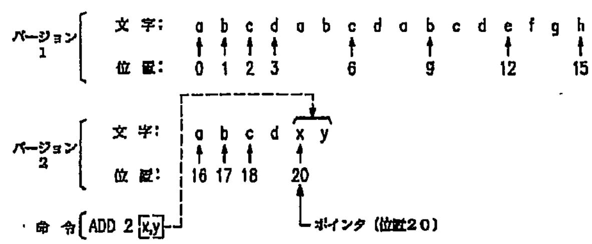
【図6】

```
1.  insert(T,seq,p,orig)
2.  {   key = seq[p,p+1,...,p+MIN-1];
3.      pos = p+orig;
4.      insert (key,pos) into T;
5.  }
```

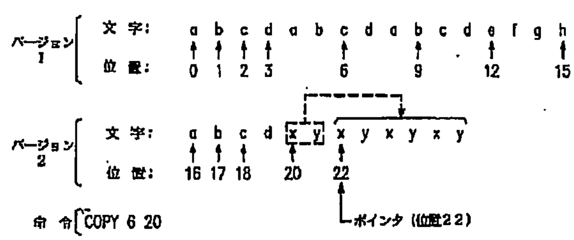
【図2A】



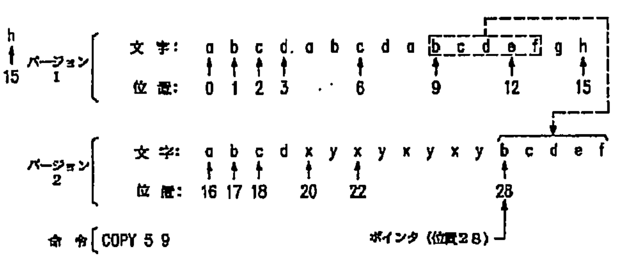
【図2C】



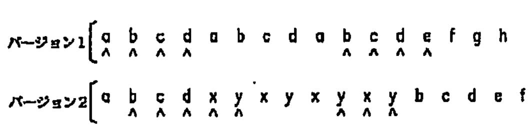
【図2D】



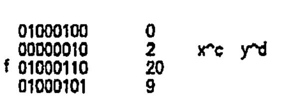
【図2E】



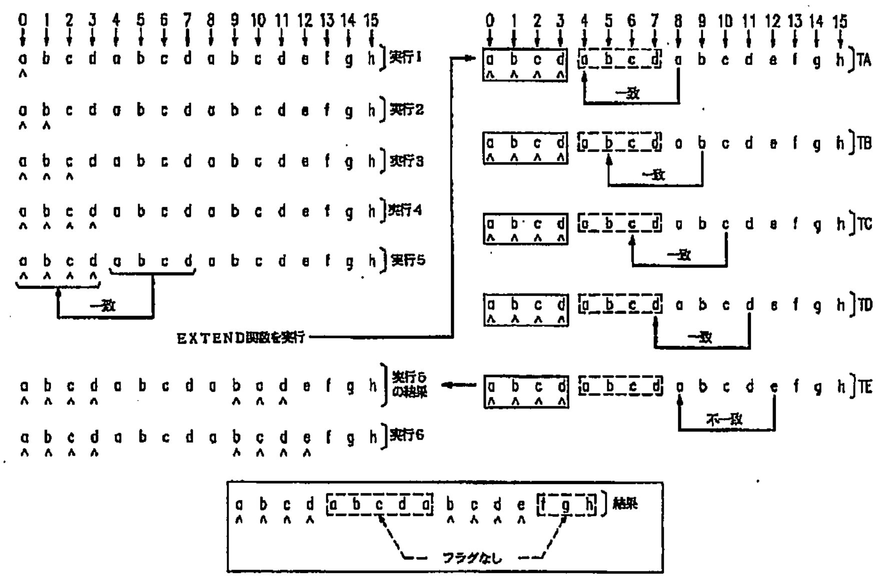
【図9】



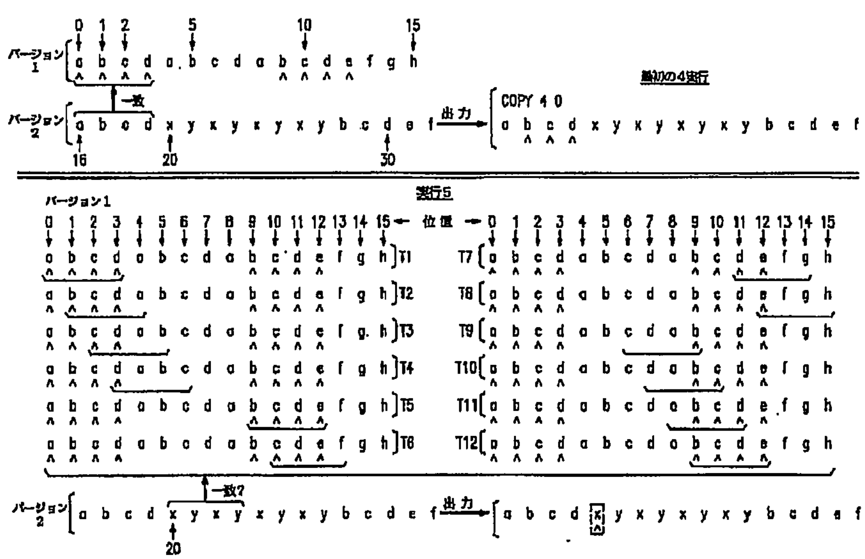
【図10】



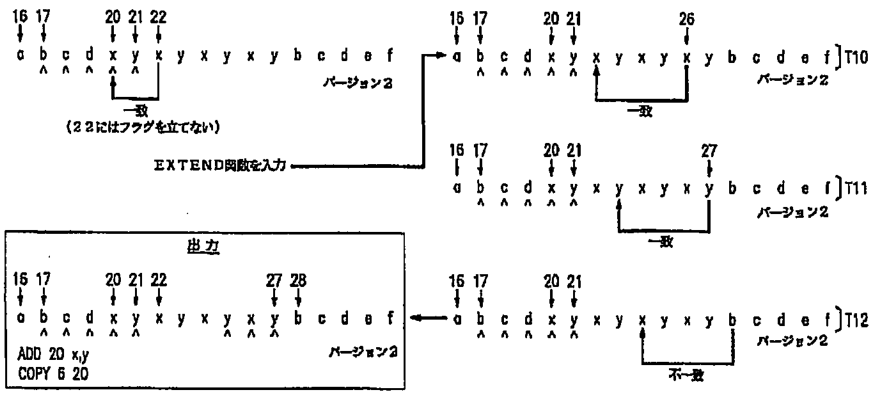
【図2F】



【図2G】



【図２Ｈ】



【図３】

```
n = length(VERSION1);
C = 0
while(true)
{   inst = readinst();
    if(inst == end-of-file)
        return;
    s = readsize();
    if(inst == ADD)
        readdata(VERSION2+c,s);
    else
    {   p = readpos();
        if(p < n)
            copy(VERSION2+c,VERSION1+p,s);
        else copy(VERSION2+c,VERSION2+p-n,s);
    }
    c = c + s;
}
```

【図５】

```
Initialize table of positions T to empty;
process(VERSION1);
process(VERSION2);
process(seq)
{   n = length(VERSION1);
    m = length(seq);
    c = 0;
    add = -1;
    pos = -1;
    len = MIN-1;
    while(1)
    {   while(1)
        {   p = search(T,seq,c,len);
            if(p < 0)
                break;
            len = extend(T,seq,c,p,len);
            pos = p;
        }
        if(pos < 0)
        {   if(add < 0)
                add = c;
            if(seq == VERSION1)
                insert(T,seq,c,0);
            else insert(T,seq,c,n);
            c = c + 1;
        }
        else
        {   if(seq == VERSION2)
                writeinst(add,c,pos,len);
            p = c+len-(MIN-1);
            while(p < c+len)
            {   if(seq == VERSION1)
                    insert(T,seq,p,0);
                else insert(T,seq,p,n);
                p = p+1
            }
            c = c + len;
            add = -1; pos = -1; len = MIN-1;
        }
        if(c >= m-MIN)
            break;
    }
    if(seq == VERSION2 and add >= 0)
        writeinst(add,m,-1,0);
}
```

【図８】

```
extend(T,seq,c,p,len)
{   n = length(VERSION1);
    if(p < n)
        str = VERSION1;
    else
    {   str = VERSION2;
        p = p-n;
    }
    m = length(seq);  i = c + len + 1;
    n = length(str);  j = p + len + 1;
    while(i < m and j < n)
        if(seq[i] != seq[j])
            break;
    return i-c;
}
```

【図7】

```
search(T,seq,c,len)
{   n = length(VERSION1);
    p = c + len - (MIN-1);
    key = seq[p,p+1,...,p+MIN-1];
    for(each entry e in T that matches key)
    {   pos = position(e);
        if(p >= n)
        {   str = VERSION2;
            q = pos-n;
        }
        else
        {   str = VERSION1;
            q = pos
        }
        d = q - (len - (MIN-1) );
        if(d >= 0)
            if(seq[c,c+1,...,p-1] == str[d,d+1,...,q-1])
                return pos - (len - (MIN-1) );
    }
    return -1;
}
```

フロントページの続き

(72)発明者　カームーフォン　ヴォー
アメリカ合衆国　07922　ニュージャーシィ，バークレイ　ハイツ，スウェンソン　サークル　80

US006199204B1 17+1

(12) **United States Patent**
Donohue

(10) **Patent No.: US 6,199,204 B1**
(45) **Date of Patent: Mar. 6, 2001**

(54) **DISTRIBUTION OF SOFTWARE UPDATES VIA A COMPUTER NETWORK**

(75) Inventor: **Seamus Donohue**, Artane (IR)

(73) Assignee: **International Business Machines Corporation**, Armonk, NY (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 0 days.

(21) Appl. No.: **09/158,704**

(22) Filed: **Sep. 22, 1998**

(30) **Foreign Application Priority Data**

Jan. 28, 1998 (GB) 9801661

(51) **Int. Cl.**[7] **G06F 9/445**

(52) **U.S. Cl.** **717/11**; 705/59

(58) **Field of Search** 717/3, 11; 709/203, 709/216–223; 707/200, 203; 705/26, 27, 57, 58, 59; 713/191, 189

(56) **References Cited**

U.S. PATENT DOCUMENTS

4,558,413	12/1985	Schmidt et al.	364/300
5,155,847	10/1992	Kirouac et al.	395/600
5,473,772	* 12/1995	Halliwell et al.	395/712
5,581,764	12/1996	Fitzgerald et al.	395/703
5,752,042	* 5/1998	Cole et al.	395/712
5,778,231	7/1998	Hoff et al.	395/705
5,797,016	8/1998	Chen et al.	395/712
5,809,251	* 9/1998	May et al.	709/223
5,845,090	* 12/1998	Collins, III et al.	709/221
5,845,293	12/1998	Veghte et al.	707/202
5,859,969	* 1/1999	Oki et al.	395/200.3
5,867,714	* 2/1999	Todd et al.	395/712
5,870,610	* 2/1999	Beyda	395/712
5,881,236	* 3/1999	Dickey	709/221
5,933,647	8/1999	Aronberg et al.	395/712
5,983,241	11/1999	Hoshino	707/203
5,991,771	11/1999	Falls et al.	707/202
5,999,740	* 12/1999	Rowley	395/712
5,999,947	12/1999	Zollinger et al.	707/203
6,006,034	12/1999	Heath et al.	395/712

FOREIGN PATENT DOCUMENTS

2 321 322	7/1998	(GB).
WO 92/22870	12/1992	(WO).
WO 94/25923	11/1994	(WO).
WO98/07085	2/1998	(WO).

OTHER PUBLICATIONS

Marimba Products "Castanet" from http://www.marimba.com/products website, p. 1.

Novadigm Fact Sheet, Novadigm's EDM from http://www/novadigm.com/cb5.htm website pp. 1 and 2, and http://www.novadigm.com/cb2.htm website, pp. 1 and 2.

"Internet Finalist: Castanet", Castanet Development Team, from http://www.zdnet.com/...ts/content/pcmg/1622/pcmg0079.html website, 1997 p. 1.

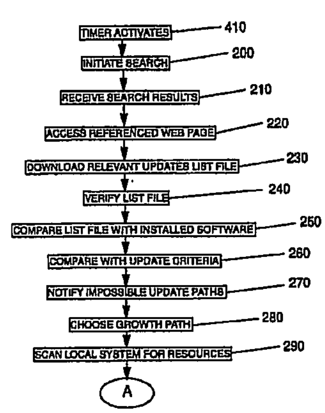
(List continued on next page.)

Primary Examiner—Kakali Chaki
(74) *Attorney, Agent, or Firm*—Jeanine S. Ray-Yarletts

(57) **ABSTRACT**

Provided is a method and mechanism for automating updating of computer programs. Conventionally, computer programs have been distributed on a recording medium for users to install on their computer systems. Each time fixes, additions and new versions for the programs were developed, a new CD or diskette was required to be delivered to users to enable them to install the update. More recently some software has been downloadable across a network, but the effort for users obtaining and installing updates and the effort for software vendors to distribute updates remains undesirable. The invention provides an updater agent which is associated with a computer program and which accesses relevant network locations and automatically downloads and installs any available updates to its associated program if those updates satisfy predefined update criteria of the updater agent. The updater agents are able to communicate with each other and so a first updater agent can request updates to programs which are prerequisites to its associated program.

9 Claims, 5 Drawing Sheets



OTHER PUBLICATIONS

"Review: Marimba Castanet 3.0", Bradley F Shimmon, from http://www/lantimes.com/testing/98aug/808c011a.html website, 1998, pp. 1–4.

Pell et al "Mission Operations with and Automous Agent" Aerospace Conference, 1998 IEEE, Mar. 21–28, 1998, vol. 2, pp. 289–313.

Yeung et al "A Multi–Agent Based Tourism Kiosk on Internet" Proceedings of the Thirty–First Hawaii Int'l Conf on System Sciences, Jan 6–8, 1998, vol. 4, pp. 452–461.

Computer Database record 02071725 of Home Office Computing, v15, n5, p66(5), May 1997, D Haskin, "Save time while you sleep (round–the–clock computing chores)".

Computer Database record 02046777 of Computer Shopper, v16, n4, p568(5), Apr. 1997, D Aubrey, "Changing channels (Internet broadcasting)".

Computer Database record 02046764 of Computer Shopper, v16, n4, p540(2), Apr. 1997, L Bailes, "First Aid 97: a good does of preventive medicine (Cybermedia First Aid 97)".

Automating Internet Service Management, "The First Distributed Object–Oriented Automation Environment for Managing Internet Services", James Herman, Northeast Consulting Resources Inc.

* cited by examiner

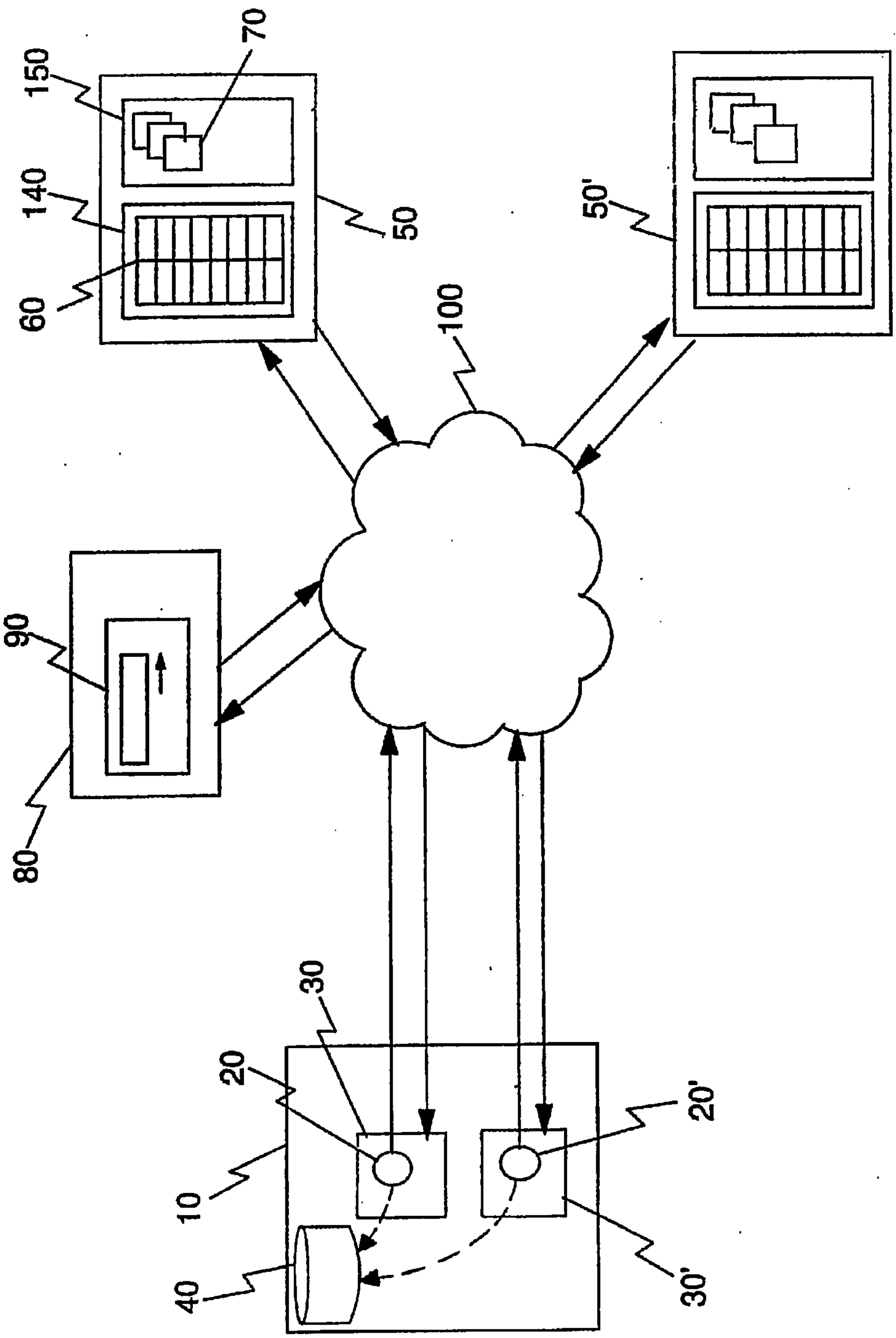


FIG. 1

120

60

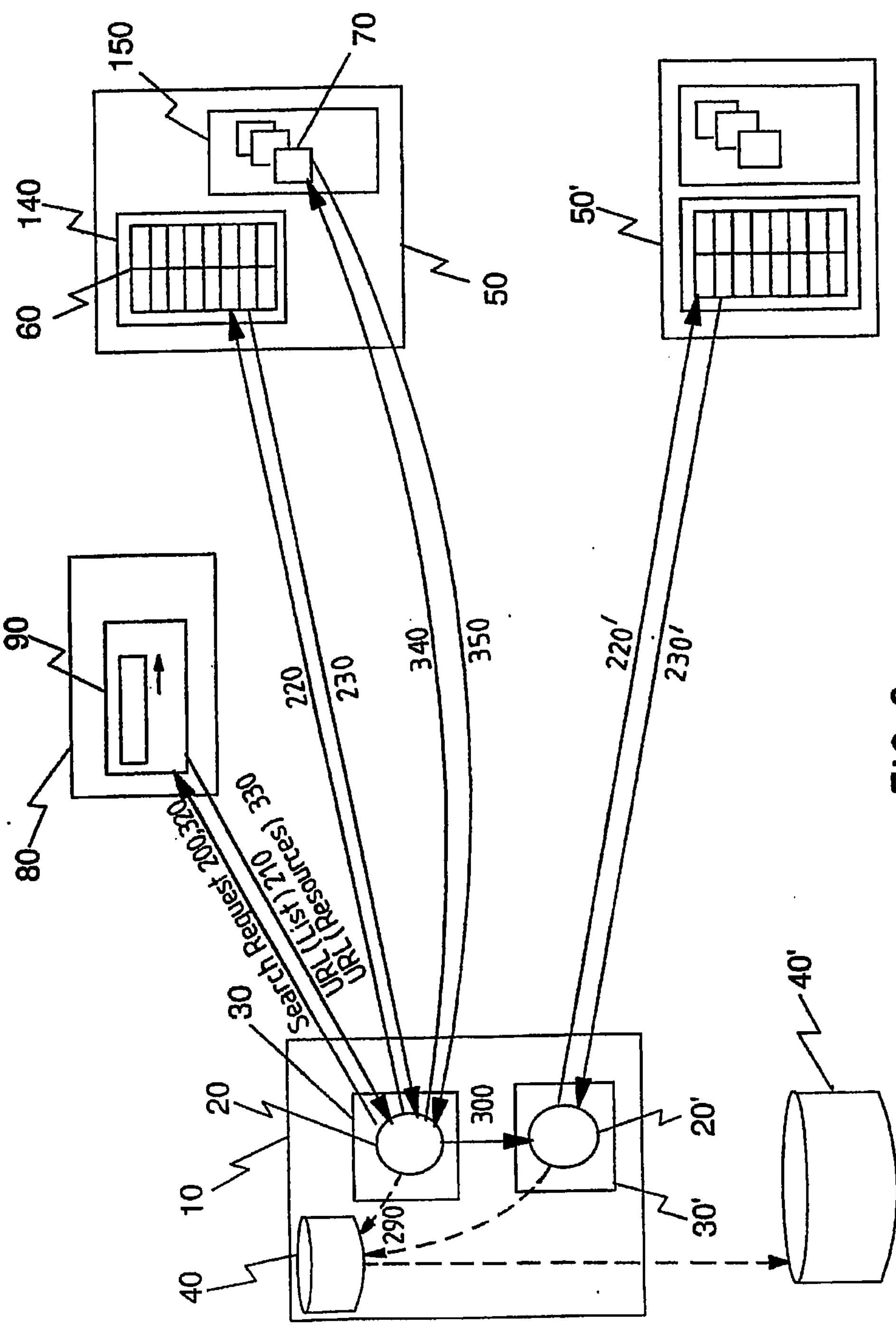
PRODUCT SET	UPDATE RESOURCES	PREREQUISITES
SOFTProd1 v1.0.0	———	OPER.SYST3 v2.0
SOFTProd1 v1.0.1	Patch1 for SOFTProd1	OPER.SYST3 v2.0
SOFTProd1 v2.0.0	Patch2 for SOFTProd1	OPER.SYST3 v2.0
SOFTProd1 v3.0.0	SOFTProd1 v3.0.0 (replacement)	OPER.SYST3 v2.0
SOFTGame2 v1.0	———	OPER.SYST3 v2.0
SOFTGame2 v2.0	Patch1 for SOFTGame2	OPER.SYST3 v3.0
SOFTGame2 v3.0	Patch2 for SOFTGame2	OPER.SYST3 v3.0

130

110

160

FIG. 2



150
70
140
60
50
50'
90
80
220
230
340
350
220'
230'
Search Request 200,320
URL (List) 210 330
URL (Resources)
30
20
10
300
290
40
20'
30'
40'

FIG. 3

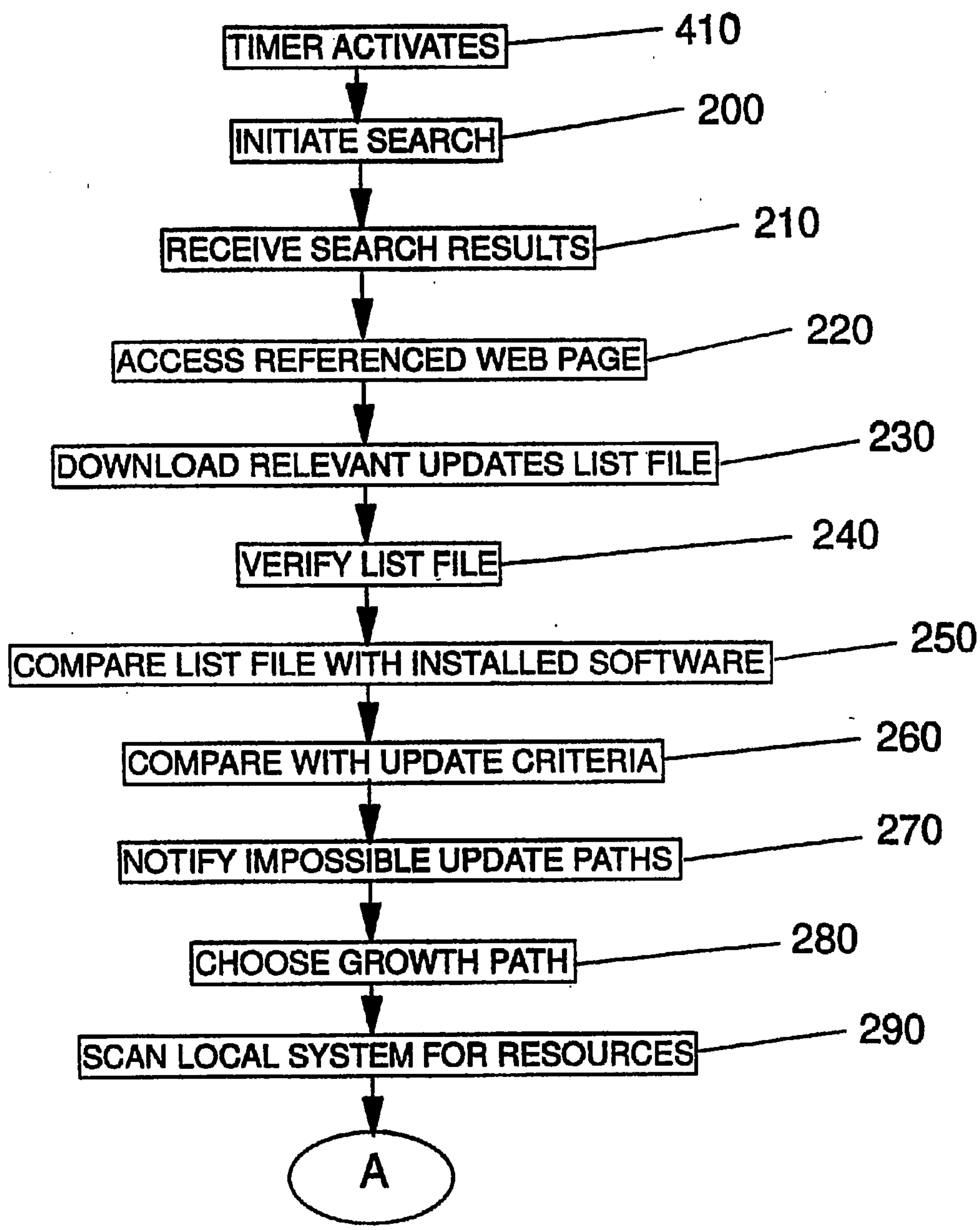

TIMER ACTIVATES
410
INITIATE SEARCH
200
RECEIVE SEARCH RESULTS
210
ACCESS REFERENCED WEB PAGE
220
DOWNLOAD RELEVANT UPDATES LIST FILE
230
VERIFY LIST FILE
240
COMPARE LIST FILE WITH INSTALLED SOFTWARE
250
COMPARE WITH UPDATE CRITERIA
260
NOTIFY IMPOSSIBLE UPDATE PATHS
270
CHOOSE GROWTH PATH
280
SCAN LOCAL SYSTEM FOR RESOURCES
290
A

FIG. 4A

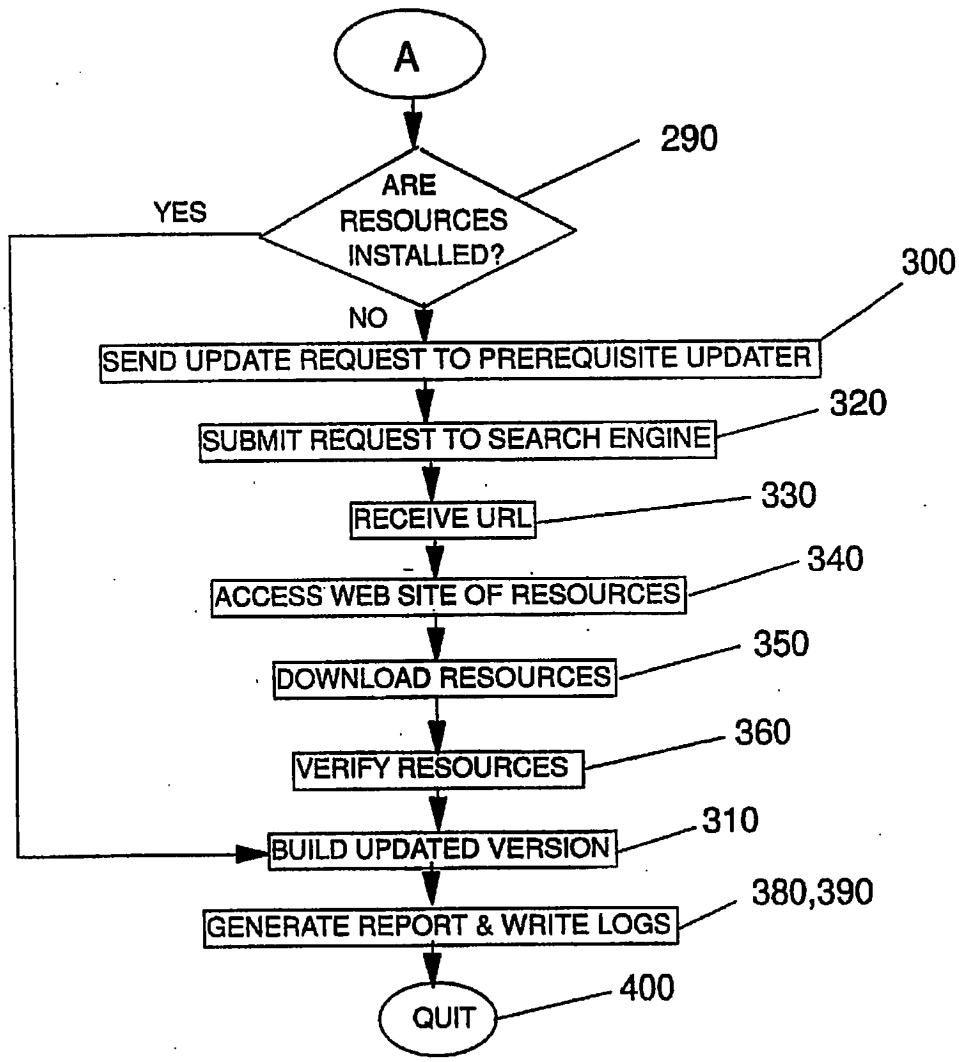

A
290
ARE RESOURCES INSTALLED?
YES
NO
300
SEND UPDATE REQUEST TO PREREQUISITE UPDATER
320
SUBMIT REQUEST TO SEARCH ENGINE
330
RECEIVE URL
340
ACCESS WEB SITE OF RESOURCES
350
DOWNLOAD RESOURCES
360
VERIFY RESOURCES
310
BUILD UPDATED VERSION
380,390
GENERATE REPORT & WRITE LOGS
400
QUIT

FIG. 4B

DISTRIBUTION OF SOFTWARE UPDATES VIA A COMPUTER NETWORK

FIELD OF INVENTION

The present invention relates to distribution of software via a computer network and to a mechanism for accessing software enhancements, corrections or new versions via a computer network. A 'network' of computers can be any number of computers that are able to exchange information with one another, and may be arranged in any configuration and using any manner of connection.

BACKGROUND

Software has conventionally been distributed in the form of programs recorded on a recording medium such as a diskette or compact disk. Customers buy the recording medium and a licence to use the software recorded on the medium, and then install the software onto their computers from the recording medium. The manufacture and distribution of the pre-recorded recording media are expensive, and this cost will be passed on to the customer. Also, the effort for customers of ordering or shopping for the software is undesirable.

The distribution cost is particularly problematic because most software products are frequently updated, both to correct bugs and to add new features, after the software has been delivered to the user. Some types of software products are updated many times each year. The cost of sending a new diskette or CD to all registered customers every time the software is upgraded or corrected is prohibitive and, although many customers want their software to be the most up-to-date, highest performance version and to be error free, not all customers want to receive every update. For example, the vendor may charge more for updates than the customer wants to spend, or new versions may require upgrading of other pre-requisite software products which the customer does not want to buy, or migrating to new versions may require migration of data which would disable the customer's system for a period of time.

Thus, software vendors tend to publicise the availability of new versions of their software and leave it for the customer to decide whether to purchase the latest upgraded version. For some software products, however, it is appropriate for the software vendor to proactively send out upgraded versions, or at least error correction and enhancement code (known as "patches") for their software products. Whatever a particular company's policy, significant costs and effort are involved in releasing these various types of software updates.

Increasingly, software distributors are using the Internet as a mechanism for publicising the availability of updates to their software, and even for distributing some software. The Internet is a network of computer networks having no single owner or controller and including large and small, public and private networks, and in which any connected computer running Internet Protocol software is, subject to security controls, capable of exchanging information with any other computer which is also connected to the Internet. This composite collection of networks which have agreed to connect to one another relies on no single transmission medium (for example, bidirectional communication can occur via satellite links, fiber-optic trunk lines, telephone lines, cable television wires and local radio links).

The World Wide Web Internet service (hereafter 'the Web') is a wide area information retrieval facility which provides access to an enormous quantity of network-

accessible information and which can provide low cost communications between Internet-connected computers. It is known for software vendors, customers who have Internet access to access the vendors' Web sites to manually check lists of the latest available versions of products and then to order the products on-line. This reduces the amount of paperwork involved in ordering software (and is equally applicable to other products). Some companies have also enabled their software to be downloaded directly from a Web site on a server computer to the customer's own computer (although this download capability is often restricted to bug fixing patches, low cost programs, and demonstration or evaluation copies of programs, for security reasons and because applying patches tends not to require any change to pre-requisite software or any data migration).

Information about the World Wide Web can be found in "Spinning the Web" by Andrew Ford (International Thomson Publishing, London 1995) and "The World Wide Web Unleashed" by John December and Neil Randall (SAMS Publishing, Indianapolis 1994). Use of the WWW is growing at an explosive rate because of its combination of flexibility, portability and ease-of-use, coupled with interactive multimedia presentation capabilities. The WWW allows any computer connected to the Internet and having the appropriate software and hardware configuration to retrieve any document that has been made available anywhere on the Internet.

This increasing usage of the Internet for ordering and distribution of software has saved costs for software vendors, but for many software products the vendor cannot just rely on all customers to access his Web pages at appropriate times and so additional update mechanisms are desirable.

As well as the problem of manufacture and distribution cost associated with distributing media, there is the problem that customers typically need to make considerable proactive effort to find out whether they have the best and the latest version and release of a software product and to obtain and apply updates. Although this effort is reduced when Internet connections are available, even a requirement for proactive checking of Web sites is undesirable to many users since it involves setting up reminders to carry out checks, finding and accessing a software provider's Web site, navigating to the Web page on which latest software versions and patches are listed, and comparing version and release numbers within this list with the installed software to determine whether a relevant product update is available and to decide whether it should be ordered. There may be an annoying delay between ordering an update and it being available for use, and even if the update can be downloaded immediately the task of migrating to an upgraded version of a software product can be difficult. If these steps have to be repeated for every application, control panel, extension, utility, and system software program installed on the system then updating becomes very tedious and time consuming. Therefore, manual updating tends not to be performed thoroughly or regularly.

There is the related problem that software vendors do not know what version of their software is being used by each customer. Even if the latest version of their software has been diligently distributed to every registered customer (by sending out CDs or by server-controlled on-line distribution), there is still no guarantee that the customer has taken the trouble to correctly install the update. This takes away some of the freedom of software developers since they generally have to maintain backward compatibility with previous versions of their software or to make other concessions for users who do not upgrade.

It is known in a client-server computing environment for a system-administrator at the server end to impose new versions of software products on end users at client systems at the administrator's discretion. However, this has only been possible where the administrator has access control for updating the client's system. This takes no account of users who do not want upgrades to be imposed on them.

Yet a further related problem is that software products often require other software products to enable them to work. For example, application programs are typically written for a specific operating system. Since specific versions of one product often require specific versions of other products, upgrading a first product without upgrading others can result in the first product not working.

"Insider Updates 2.0" is a commercially available software updater utility from Insider Software Corporation which, when triggered by the user, creates an inventory of installed software on a user's Apple Macintosh computer and compares this with a database of available software update patches (but not upgraded product versions) and downloads relevant updates. "Insider Updates" shifts the responsibility for finding relevant updates from the user to the database maintainer, but the access to update patches is limited to a connection to an individual database and the tasks of scanning the Internet and on-line services to find updates and of maintaining the database of available updates require significant proactive effort. "Insider Updates" does not install the updates or modify the user's software in any way. "Insider Updates" does not address the problem of unsynchronised prerequisite software products.

A similar product which scans selected volumes of a computer system to determine the installed software and which connects to a database of software titles for the Apple Macintosh, but does not download updates, is Symmetry Software Corporation's "Version Master 1.5".

An alternative update approach is provided by "Shaman Update Server 1.1.6" from Shaman Corporation, which consists of: a CD-ROM (updated and distributed monthly) that users install on a PowerMac file server; client software for each Macintosh computer to be inventoried and updated; and means for accessing an FTP site storing a library of current updates. "Shaman Update Server" creates an inventory of networked computers and downloads and distributes latest versions of software to each computer. Network administrators centrally control this inventory and updating process. The distribution of CD-ROMs has the expense problems described earlier.

SUMMARY OF INVENTION

According to a first aspect of the invention, there is provided an updater component for use in updating one or more computer programs installed on a computer system connected within a computer network, the updater component including:

- information for identifying one or more locations within the network where one or more required software resources are located;
- means for initiating access to the one or more locations to retrieve the one or more required software resources; and
- means for applying a software update to one of said installed computer programs using the one or more retrieved software resources.

An updater component according to the invention preferably controls upgrading of, and fixing of bugs within, an associated software product or products automatically without requiring any interaction by the user after an initial agreement of update criteria. The update criteria can be associated with the products' licensing terms and conditions. This ensures that users who adopt a suitable update policy can always have the most up-to-date software available, with errors being corrected automatically from the viewpoint of the user. The user does not need to know where software updates come from, how to obtain them or how to install them since the update component takes care of this. The software vendor avoids having to ship special CDs or diskettes to correct errors or provide additional features; the vendor can easily release code on an incremental basis such that customers receive new product features sooner and with no effort.

An updater component according to a preferred embodiment of the invention performs a comparison between available software updates and installed software on the local computer system to identify which are relevant to the installed software, compares the available relevant updates with update criteria held on the local computer system (these update criteria are predefined for the current system or system user), and then automatically downloads and applies software updates which satisfy the predefined criteria.

This automatic applying of software updates preferably involves installing available software patches and/or upgraded versions in accordance with both the predefined update criteria and instructions for installation which are downloaded together with the program code required for the update. This feature of executing dynamically downloaded instructions provides flexibility in relation to the types of updates that can be handled by an updater component. It can also be used to enable a single generic updater component to be used with many different software products. Alternatively, the installation instructions for certain software updates may be pre-coded within the updater component. The "software resources" are typically a combination of program code, machine readable installation instructions and any required data changes such as address information.

The information for use in identifying a network location may be explicit network location information or it may be a software vendor name or any other information which can be used as a search parameter for identifying the location. In the preferred embodiment, the information is a product identifier which is provided by the updater component to a search engine to initiate a search to identify the relevant network location at which are stored the software resources for implementing updates to that product. This search may be performed by a conventional Internet (or other network) search engine which is called by the updater component. When the search engine returns an identification of the network location, the updater component retrieves from this location a list of available relevant updates, checks the list against the locally held software product version and against predefined update criteria, and retrieves the update resources onto the local computer system if those criteria are satisfied.

According to a preferred embodiment of the invention, a standardised naming convention is used for software resources from which to build software updates, and the updater component can search for these resources on a Network Operating System filesystem. This allows software resources to be stored at multiple locations to mitigate against network availability problems and makes it easier for developers and distributors to provide their error-fixing patches and upgraded versions of software products. For example, a developer can make new software updates available via a public network disk drive on their LAN using a known filename or via a published Uniform Resource Locator (URL) which can be searched for using known key words.

Updater components are preferably an integral part of the products they will serve to update. Hence, the updater component is distributed to software users together with an initial version of a software product, the updater component then automatically obtaining and applying software updates in accordance with preset criteria (such as a time period between successive searches for updates, and whether the particular user has selected to receive all updates or only certain updates—such as to receive updating patches but not replacement product versions for example).

The updater component's update capability preferably includes updating itself. Indeed, the update criteria may be set such that the updater component always accesses appropriate network locations to obtain updates to itself before it searches for software resources for updating its associated software products.

An updater component according to the invention preferably includes means for checking whether pre-requisite products are available, and are synchronised to the required version, as part of the process of selecting an update path for the current product. In a preferred embodiment, as well as checking their availability, the updater component is capable of instructing the updater components associated with pre-requisite software products to initiate updates to their software where this is the agreed update policy. If each software product's updater component is capable of triggering updates to pre-requisite products, then updates can ripple through the set of installed software products without the user having to be involved in or aware of the updates. This capability is a significant advantage over prior art updater agents which do not deal with the problem of unsynchronised software versions when one updates, and supports the increasing trend within the software industry for collaboration between distributed objects to perform tasks for the end user.

The updater component preferably also includes a mechanism for verifying the authenticity of downloaded software, using cryptographic algorithms. This avoids the need for dedicated, password-protected or otherwise protected software resource repository sites. The software resources can be anywhere on the network as long as they are correctly named and or posted to the network search engines.

Thus, the present invention provides an agent and a method for obtaining and applying software updates which significantly reduces the cost and effort for software distributors of distributing and tracking software updates and significantly reduces the effort for system administrators and end users of applying updates to installed software.

BRIEF DESCRIPTION OF DRAWINGS

The present invention will now be described in more detail, by way of example only, with reference to the accompanying drawings in which:

FIG. 1 is a schematic representation of a computer network including a local computer system having an installed updater component, server computers storing lists of available updates and storing software resources for applying updates, and a search engine for locating the servers;

FIG. 2 is an example of a software vendor's list of their software versions and the resources and prerequisites for building from one version to another;

FIG. 3 represents the sequence of operations of an updater component according to an embodiment of the invention; and

FIG. 4 is a further representation of the sequence of operations of an updater component.

DETAILED DESCRIPTION OF PREFERRED EMBODIMENTS

As shown in FIG. 1, an updater component **20** is installed in system memory of a conventional network-connected computer system **10**, together with an associated computer program **30**. The updater component may have been delivered to the user of the local computer system on a storage medium (diskette or CD) for him to install, or it may have been explicitly downloaded from another computer system. In preferred embodiments of the invention, updater components are integrated within the computer program they are intended to maintain (or are otherwise delivered via the same mechanism and at the same time as their associated program). The updater component is then installed as part of the installation procedure of its associated program, such that the user is not required to take any special action to obtain or activate it. The installation of each updater component includes the updater component registering itself with the operating system (more generally, updaters register with a repository **40**, which may be central or distributed), such that at least the updater components on the local system are identifiable and contactable by address information, and/or their product identifier, within the register entry.

It is a feature of the preferred embodiment of the invention that each updater component can locate, can be located by, and can communicate with other updater components which manage other software products. This capability is used when one updater component requires another one to update to a specific level before the former can execute its own update, as will be discussed below. This is enabled by the updater components registering within the operating system or other repository **40**.

In the preferred embodiment, each registration entry contains two items: the updater path and the updater network address. The path is the location of the updater component binary file so that the updater component can be launched by the operating system during the boot up process. This ensures that the updater component is always active and ready to perform work or handle requests issued to it from other updater components. The network address is the address used by components on other computer systems in the network to locate it on the network and to communicate with it.

An example of such registration using a UNIX (TM) operating system and the TCP/IP protocol suite uses the following naming convention for updater components: SoftwareVendorName+_product_name+_updater.

Path registrations can be entered in the UNIX/etc/inittab file to store the path entry. When, for example, the updater component for IBM Corporation's DB2 (TM) database product is installed it will add an entry to the /etc/inittab file of the form:

ibm_db2pe_updater:2:respawn:/usr/abin/db2_updater_binary

Every time the computer system reboots it will read this file and launch the DB2 updater component. The "respawn" keyword in the updater entry ensures that, should the updater component process fail for some reason during general system operation, it will be restarted by the operating system automatically. This approach will ensure that all updater components for all installed applications are always active.

Network Location registrations can be entered in the UNIX/etc/services file. For example when the DB2 updater component is installed it will add an entry to the /etc/services file of the form:

ibm_db2pe_updater 5000/tcp #net location of DB2 updater component

When another updater component wishes to communicate with the DB2 updater component it will find it by searching this file for the DB2 updater component name ibm_db2pe_updater (actually done indirectly by the UNIX call getservbyname()—the name is built by the caller according to the standard naming convention). When it is found it knows that the DB2 updater is listening for connections on port number 5000 and will use the TCP protocol. This allows the updater component in question to establish a link to the DB2 updater component and start a conversation (described later).

For an updater component to find and talk to another updater component on another remote machine the above information would have to be augmented by having a repository 40' which is accessible from both machines (preferably a central or distributed database accessible from anywhere in the network, such as a Web Page or pan-network file) and is available to all updater components that require it. Entries would be of the form updater_name machine_ip_address (OR DNS entry), port number, protocol.

For example, the manufacturing department of an organisation may have three computer systems on which distributed software products collaborate with each other, the systems being called a, b and c. Typical entries in the Web page or file manufacturing_collaborators.html might be:

ibm_catia_updater a.manufacturing.com 5000 tcp
ibm_db2pe_updater b.manufacturing.com 5100 tcp
ibm_cics_updater c.manufacturing.com 4780 tcp

An updater component can then connect and talk to any other updater component using the DNS name to create an IP address and the port number which the remote updater component is listening to at that address.

The steps of updater registration at installation are therefore:

1) Create entry in /etc/inittab file (register updater process code location)
2) Create entry in /etc/services file (register updater process local address)
3) Create entry in central database file (register updater process pan-network address).

The installation process may also involve providing to the updater component the local IP address of a Web proxy server. It will be clear to persons skilled in the art that many alternative registration implementations are possible.

Updater components include data fields for an identifier and version number for their associated software products. The updater components may be delivered to customers with these fields set to null values, and then the installation procedure includes an initial step of the updater component interrogating its software product to obtain an identifier and current program version and release number. Alternatively, the software vendor may pre-code the relevant product ID and version number into the updater component.

The system 10 of FIG. 1 is shown connected within a network 100 of computers including a number of remote server systems (50,50') from which software resources are available for applying updates to programs installed on the local system 10. Each server system includes within storage a list 60 of the latest versions of, and patches for, software products which are available from that server. Each vendor is assumed here to make available via their Web sites such a list 60 of software updates (an example of which is shown in FIG. 2) comprising their product release history, in a format which is readable by updater components, and to make available the software resources 70 required to build the releases from a given level to a new level (this transition from a software product release to a new level will be referred to hereafter as a 'growth' path). The entries in the software updates list 60 include for each software product version 110 an identification 120 of the software resources required for applying the update and an identification 130 of its prerequisite software products and their version numbers. In some cases, the required resources are complete replacement versions of software and associated installation instructions. In other cases, the resources comprise patch code for modifying an existing program (e.g. for error correction) and the patch's installation instructions.

For the current example, we will assume that the network 100 is the Internet, although the invention may be implemented within any computer network. Also shown within the network 100 is a server system 80 on which a search engine 90 is installed for use in finding update source locations on the network. This is shown located remotely from the local system 10, although it need not be. In the Figure, each updater component 20 is shown associated with a single program 30, and it is a feature of this embodiment of the invention that all installed software products have associated updater components which manage them, but neither of these features is essential to the invention as will be explained later.

The operation of an updater component will now be described, with reference to FIGS. 3 and 4. When an installed updater component executes, its first action is to initiate 200 a search for available updates to the particular software product, providing to one or more search engines 90 as search arguments the product identifier and product version release number obtained at install time. Assuming that software vendors provide via their Web sites a list 60 of available product updates referenced by product identifier and release number 110 (or some other consistent naming convention is used), the search should identify the relevant Web site 140 on which update information is available. If the initial attempt to start a search engine is unsuccessful, then the updater component will attempt to start a different search engine (which may be in a different geographical location to the first), but could alternatively wait for a preset time period and then retry. A URL identifying the relevant Web site 140 for update information is returned 210 to the updater component as a result of the search.

The updater component uses the URL to access 220 the list 60 and downloads 230 a file 160 comprising the portion of the list 60 of available updates which relates to the particular product. The updater component then performs steps 240–280 as shown in FIG. 4. Each file 160 contains message digests (e.g. MD5) which are digitally signed. The retrieved file 160 is then analyzed 240 using a digital signature checking algorithm (such as the algorithm described in U.S. Pat. No. 5,231,668). This is important to verify that the file 160 represents the correct software updates list for the particular software product, and that the file has not been tampered with since signing. Also, checking for the digital signature is a useful way of filtering the results of the search since these may include a plurality of Web page URLs other than the correct one (the search may find other pages which have a reference to the named product version, including pages not published by the software vendor). If an attempt to download and verify a file is not successful, then the updater component moves on to the next URL found in the search.

The updater component then performs on the local computer system a comparison 250 between the current installed software product's identifier and release number and the listed available updates in the retrieved file 160. This com-

parison determines possible growth paths from the current to updated versions, but these possible growth paths are then compared 260 with predefined update criteria, and any possible paths which do not satisfy the update criteria are discarded. Thus, the updater component determines whether it is possible to migrate from a current software product to the available new versions and whether it is possible to apply patches to the current version under the currently agreed licence terms and conditions.

For example, the software product licence may enable migration to any future version of the product and application of any available patches, or only migration up to a specified version, or it may only permit applying of available patches which modify or correct errors in the current version. Possible update paths which are unavailable due to current license limitations are notified 270 as a system generated message sent to the software asset manager (who may be an end user or IT procurement manager) of the currently installed version, to enable them to make decisions about whether the current licence is adequate.

As well as licence restrictions as to the updates that are possible, an updater component's update criteria or growth policy includes a cycle period (for example weekly or monthly) and criteria for determining which of a plurality of possible growth paths to select (such as always select latest version permitted by licence, or always select latest patch and only notify availability of new versions, or only select new versions if prerequisite software is already available on local system). The growth criteria may also include control information such as when to upgrade to new versions that are downloaded by the updater component—if data migration is required when migrating to a new software product version it may be essential for this to be done outside of office hours or only at a single scheduled time each month or each year and this can be controlled by the updater component.

The growth policy definition may also include a parameter determining that updating of pre-requisite software products should be requested when required to maintain synchronisation with the current product. This will be described in more detail below. Persons skilled in the art will appreciate that there is great flexibility in the criteria that can be set and applied by the updater component.

The updater component then decides 280 on a particular growth path (i.e. which available version to upgrade to) from the set of possible growth paths using the update criteria. For example, the updater component may select the highest possible version or release number of the available updates which is permitted by the update criteria, if that is the update policy.

The updater component performs 290 (see FIGS. 3 and 4) a scan of the operating system file system to check whether the required software resources are already available on the local computer system. The required resources are the software update artifacts required to bring the current application software to the new level, and the software updates required for updating pre-requisite software to required levels. Each updater component associated with pre-requisite installed products is contacted 300 to ensure (a) that it is installed, and (b) that it is at or greater than the required pre-requisite level. If all required resources are available locally or on another machine (in the case of software relying on some pre-requisite software operating on a remote machine), and have been verified, then the updater component progresses to the step 310 (see FIG. 4) of building the updated software version. If not, the update component must obtain the required resources.

As shown in FIGS. 3 and 4, if required software resources for building the updated version are not found on the local system, the updater component submits 320 a further request to one or more search engines to find the required resources. The search engine returns 330 one or more URLs and the updater component uses these to retrieve 340,350 the software resources into storage of the local computer system. At this stage, the updater component or the user need not have any knowledge of what corrections or enhancements may be included in the new version—the update criteria determine what type of updates are required such that the user is spared the effort of studying the content of every update. In practice, it is desirable for users to be able to determine the effects of updates and so the software resources for the update include a description of these effects which a user or administrator can read.

As examples, the software product to be updated may be a word processor application program. If the word processor as sold missed certain fonts or did not include a thesaurus, patches may subsequently be made available for adding these features. The updater component has the capability to add these to the word processor, subject to the update criteria.

In alternative embodiments of the invention, the search for required software resources is unnecessary following the initial search for the updates list (or is only necessary where there are pre-requisite software products as well as patches or new versions for the current product—see below). This is because the update software resources required directly by the current product are stored in association with the list of required resources. That is, the list includes a pointer to the network location of the required resources such that a selection of a growth path from the list involves a selection of a pointer to the network location of the required updates (and possibly also pointers to the locations of pre-requisite software products).

A second verification by digital signature checking is performed 360 (see FIG. 4), this time on the downloaded resources. After verifying 360 the legitimacy of the downloaded resources, the updater component automatically builds 310 the installation in the target environment in accordance with the update policy. In practice, this may require information from the user such as an administration password, or a database usage parameter value, but in the preferred embodiment of the invention installing of the downloaded code is automatic in the sense that it does not require the user to know or obtain from elsewhere any installation information and in that it generally enables the user to be freed from making any decisions at run time if the predefined update criteria enable the updater component to automatically apply updates.

It is well known to include machine readable installation instructions encoded in a shell (for example as Script, or an interpretive language such as PERL, or an executable such as setup.exe in the case of applications on Microsoft's Windows (TM) operating system). Updater components according to the invention will download 350 the machine readable instructions together with the relevant software resources and will automatically execute them 310. The updater component thus automatically processes installation instructions, avoiding the input from a person which is conventionally required. The Scripts can be adapted to reuse information gleaned from the first human installer who installed the first version of the updater component (for example, information such as user name and password of application administrator, installation directory, etc).

The method of updating according to the preferred embodiment of the invention requires software vendors to

organise the software resources required to build from one product level to another. For example, a move from version 1.1.1 to 1.1.4 would typically include a series of patches to be applied, and the required order of installation if any would advantageously be encoded in machine processable installation instructions. The user is then spared the effort and the risk of human error which are inherent in methods which require the user to control the order of application of fixes and enhancements. The problem of how to migrate from one product level to another is thus dealt with by the software vendor instead of the customer, and updater components can only move to levels supported by the vendor (i.e. those growth paths published by the software vendor for a specific existing product level).

The updater generates **380** a report and writes **390** to log records, and then quits execution **400** (in the preferred embodiment the updater goes into a sleep or idle state) until activated again **410** upon expiry of a predetermined update cycle period (the repeat period parameter is configured when the updater component is installed).

Structure of Updater Component

The structure of an updater component comprises data, methods for operating on that data, and a public application programming interface (API) which allows other updater components to contact and communicate with it. This structure will now be described in detail.

UPDATER COMPONENT DATA:

The updater component includes the following persistent data:

Product_ID: an identifier of the software product which is managed by this updater component

Current_Installed_Version: a version identifier for the installed software (e.g. version 3.1.0)

Current_License: a version identifier corresponding to the software product version up to which the current software license allows the user to upgrade (e.g. version 4.0.z). Alternatively, this may be a licence identifier (e.g. LIC1) for use when accessing machine readable licence terms.

Installation_Environment:

a list of attribute name/attribute value pairs.

This is used by the updater component to store values entered by the user when the updater was used for the first time. For example, the updater installation userid and password, possibly the root password, the installation directory, the web-proxy server address, search engine URLS, log file name, software asset manager e-mail address etc. This data will be re-used when subsequent automatic updates are required.

Growth policy parameters:

a. Growth_Cycle: data determining whether the updater component should attempt to update its software product every day, week or month, etc.

b. Growth_Type: data determining whether the updating is limited to bug fixing and enhancements (i.e. patches) only or requires upgrading to the latest release in each growth cycle.

c. Force_Growth: (YES/NO) a parameter determining whether to force other software resources to upgrade if that is a pre-requisite for this software to upgrade. (Some implementations will provide more flexible controls over forcing other software to update than this simple YES/No)

Last_Growth_Time: Date and time when updater component last executed

The updater component also includes the following non persistent data:

Possible_Growth_Paths:

transient data representing the available upgrade paths (e.g. version numbers 3.1.d, 3.2.e, 4.0.a)

PRIVATE UPDATER FUNCTIONS:

The updater component logic includes the following methods:

Discover_Possible_Growth_Paths()

Search for Growth_Path information for this software product on the Internet (or Intranet or other network). This search method initiates a search via a standard search engine server. The information returned is a list of newer versions and associated pre-requisite product information.

The Growth_Path information is then reduced in accordance with the Growth policy parameters. For all members in the Growth_Paths list, a check is performed of whether appropriate versions of pre-requisite products are available on the local and/or remote computer. The updater components managing these pre-requisite products are accessed and forced to grow if this is the policy.

If all pre-requisite products exist locally at the correct level, or are available remotely on the network and there is with a "force growth" policy, then identifiers for newer versions of the software product are added to the Possible_Growth_Paths list.

Decide_Growth_Path()

Interpret the growth policy and select a single growth path. Some implementations of the invention will involve user interaction to select the path, for example if there are considerations such as whether to force updates to other programs.

Get_Resources(Parameter: Chosen_Growth_Path)

Given Chosen_Growth_Path (e.g.3.2.0), search for required resources (Parameters Product_ID, Current_Installed_Version, Chosen_Growth_Path), download all resources to local computer. This will include software required for the new version plus machine processable installation instructions.

Install_Resources()

Process installation instructions including installing required files in correct locations, possibly compilation of the files and modifying the configuration of the existing system to accommodate the software, logging all actions to a file (and enabling an "uninstall" method to undo all actions).

Grow()

Initiates methods:

Discover_Possible_Growth_Paths()

if no possible growth paths exist then updater component becomes idle else

Decide_Growth_Path()

Get_Resources(Parameter: Chosen_Growth_Path)

Install_Resources().

Then Growo writes all completed actions to log and finishes execution of the updater component. The updater component becomes idle either until time to check again for new update requirements or until prompted by another updater component to do so.

PUBLIC UPDATER COMPONENT API:

The updater component includes the following public API. These functions would be callable using existing network communications software, such as remote procedure calls, message oriented middleware, ORB (Object request broker), etc.

Get_Release()

This function is called by other updater components and returns the release level of the product managed by this updater component.

Update(new_level)

Other updater components call this function to move the product managed by this updater component to a new level indicated by the new_level parameter value. This will call the private function Grow().

Receive_Event(event details)

When an updater component receives a request to update, it must inform the calling updater component when it has completed the update or otherwise e.g. if it failed for some reason. The updater component performing the update on behalf of another updater component will call this function of the requesting updater component to communicate success of the update or otherwise. Event details can be a string like "product id, new release level, ok" or "product id, new release level, failure".

The automatic handling of the potential problem of unsynchronised pre-requisite products by enabling forcing of updates (or, if forcing of updates is not part of the update policy, sending of notifications to the software asset manager) is a significant advance over prior art update schemes.

Since the updates list file **160** returned to the updater component in response to an initial search includes an identification **130** of pre-requisite software, that information enables the aforementioned examination **290** of the updater component registration database **40,40'** to check whether pre-requisite software is available locally or remotely. If it finds all the updater components located locally or remotely, it can be sure that the software pre-requisites are available and it next needs to contact each updater component for each software product to be sure all pre-requisites are at the correct level. If an updater component **20'** having a required product identifier for pre-requisite software **30'** but not having the required version number is found locally or remotely, and if forcing of updates is the update policy, then the updater component **20** of the first computer program contacts **300** this pre-requisite updater component **20'** and requests that it attempt to update its associated pre-requisite software product **30'**. This updater component **20'** can, if necessary, request other updater components of its pre-requisite software to update their versions, and so on.

If at some stage no relevant updater component is found locally or remotely, then a message is sent to the asset manager to inform him/her of the requirement for a new product in order to grow the associated product further. If at some stage during the chain of updater requests to grow to a new level one updater component fails to move to the required level then this failure is reported back to its calling updater component, prompting failure of that components update operation, and so on back to the updater component which initiated the whole transaction.

Thus, as well as their autonomous behaviour defined by their update criteria, updater components can react to external stimuli such as requests from other updater components.

Example of Update Synchronisation

An example of the implementation of update synchronisation between two products will now be described. This example shows how one updater component can communicate with another to synchronise pre-requisite software so that all products are present and at compatible release levels.

A CORBA (Common Object Request Broker Architecture) ORB (Object Request Broker) is used for location of and communication between two updater components. Using the above public API it is a simple matter for those familiar with the art of CORBA programming to develop communication code so that one updater component can talk to another updater component anywhere on a network. In this example the component updater registration database **40** is a directory or folder available over the network (e.g. via NFS) which contains for each installed updater component a file called "updater-component_name.iop" (iop stands for interoperable object reference).

This file contains a sequence of bytes which can be converted into a reference to the updater component by any updater component which reads the file using for example the CORBA function:

CORBA::Object::_string_to_object() in C++

Furthermore this reference can be to an updater component anywhere on the network as it represents a unique address for the corresponding updater component. When updater component A has manufactured a reference to updater component B then updater component A can call a public API function simply by using, for example, a C++ mapping A→Get_Release() which will then return the value of the release level of the software managed by the A updater component.

In this example we will consider two products—IBM Corporation's DB2 database product and a Query Tool called "Query Builder", on different machines M and N respectively. (Machines M and N could be the same machine; the present example merely shows that they may also be separate). Both products have updater components which use a CORBA ORB architecture as briefly outlined above. An ORB communication daemon is active on participating systems M and N.

Step 1) Registration Phase:

The DB2 Updater Component starts when the operating system starts on system M and immediately creates a file called ibm_db2_updater.iop (according to some naming standard used to aid subsequent searches for the file) in the network file system folder or directory. This directory could be hosted on any machine and not necessarily M or N. The file contains a series of bytes which can be used to manufacture a reference to the updater component.

```
[pseudocode]
Filehandle=open("/network/filesystem/directory", "ibm_
    db2_updater.iop");
ReferenceBytes=CORBA::Object::_object_to_string();
Write(FileHandle, ReferenceBytes);
close(Filehandle);
```

QueryBuilder Updater component starts and writes its registration to the same directory or folder, again in this case calling the file ibm_querybuilder_updater.iop.

At this stage both updater components are active and have registered their presence and location in the network directory.

Step 2).

QueryBuilder attempts to grow from version 1 to version 2 but a prerequisite is DB2 version 2.1 or higher. The following sequence of actions will occur. QueryBuilder is denoted QB and DB2 as DB2.

QB: searches for file ibm_db2_updater.iop (file name manufactured according to standard) in network directory. It finds the file, reads it and converts it to a usable reference.

```
[pseudocode]
if (dbref=CORBA::Object::_string_to_object(readfile
    (ibm_db2_updater.iop)))
  then SUCCESS we have connected to the updater else
  FAIL: Prerequisite software does not exist in set of
    collaborating systems—send e-mail to software asset
    manager to notify situation.
```

Give up on trying to grow to new version.
endif.
Step 3).

At this stage we know that DB2 exists somewhere in our set of networked computers. Now we need to know if it is at the right level. We simply do this by executing its public API function Get_Release() defined above, from within the QB updater, the QB updater is therefore a client requesting the DB2 updater to do something for it, i.e. tell it what release it is.

[pseudocode]
db2_release=dbref→Get_Release();

Let us say this returns the value "2.0".

Step 4)
Client Side:

The QB Updater Component knows that this is not sufficient , it requires version 2.1. It examines its Force_Growth parameter which is, for example, "YES" meaning it should force pre-requisite software to grow to the level required before it can perform its own update procedure. Therefore the QB updater tells the DB2 updater to grow to the new release, and then waits until the pre-requisite has grown to the new release or failed in doing so.

[pseudocode]
dbref→Update("2.1", QBref); // QBref is a ready made reference to the // QB Updater. It is passed to the DB2 updater so that // it can quickly send the results, success or failure, // when the DB2 Updater has finished trying to update // itself.
EVENT=null;
While (EVENT equals null)
{do nothing;}
if (EVENT equals "SUCCESS")
then attempt to grow software managed by this updater component i.e. Query Builder.
else
Write failure to log;
do not attempt to grow;
go to sleep and try later;
endif.

Server Side:

The DB2 Updater component receives the request to grow. Which it attempts to do.

It reports the result to the calling client (it knows how to contact the calling client as it is receives a reference to the caller in the function call.)

[pseudocode]
DB2 attempts to grow.
if Growth Successful then
QBRef→Receive_Event("SUCCESS"); // Note the implementation of the // function Receive_Event simply sets the variable // called EVENT in the QB Updater component to the // value of the parameter passed in the API call , i.e. // "SUCCESS" if in this section of the IF statement.
else
QBREF→Receive_Event("FAILURE");
end if

As noted previously, predefined update criteria may determine which of an available set of updates should be applied and which should be disregarded. The update criteria may include an instruction to the updater component to send a notification to the end user or system administrator when a software update is identified as being available but applying this update is not within the update policy or is impossible. One of the examples given previously is that the update policy may be not to install full replacement versions of software products since that may require upgrading of pre-requisite software products or migration of data (for example if the software product is a database product), whereas it may be intended policy to install any error-correction patches. Notification rather than automatic installation of updates may also be implemented where to upgrade one product to a new version would require upgrading of other pre-requisite complementary products.

The update policy can also determine the degree of automation of the updating process, by defining the circumstances in which the updater requests input from the user or administrator.

The execution of a particular example updater component will now be described in more detail by way of example. This updater component's function is to keep an installed product called "Test" totally up-to-date with all released patches, but not to install replacement versions of Test. Firstly, the updater component is configured with the following data instantiations:

Product_ID: Test
Current_Installed_Version: 1.0.a
Current_License: LIC1
Installation_Environment:"USERID:TestOwner,
USERPASSWORD:easy"
"INSTALLPATH: /usr/bin/testapp/"
Growth_Cycle: weekly
Growth_Type: patches, latest, automatically
Force_Growth: no
Last_Growth_Time: Monday Aug. 10, 1997.

The updater then executes weekly, for example each Monday night at 3 am (it is the system administrator who decides the timing).

The following represents a possible execution trace for this example updater component.

Example Execution Trace

Step 1) The Growth Cycle Starts:
>>>> START: Discover_possible_Growth_Paths()
* Execute search on remote search engine (e.g. Internet Search Engine) using Phrase ("IBM Test 1.0.a Growth Paths")
Search returns URL published by software vendor outlining current growth paths for product;
* Download URL:
File contents are:
"1.0.b,none; 2.0, other_required product_product_id 1.0.c;"
* Authenticate URL file using hashing algorithm and digital signature.
If not authentic, return to search for another URL matching criteria
* Build growth_path_list: growth_path list="1.0.b, none;
2.0, other_required_product_id 1.0.c;"
* Remove all but patch level increases (according to Growth_Policy) from Growth_path list (i.e. only those with the first version and second release number matching 1.0).
* growth_path list="1.0.b, none;"
* For all members in list, ensure prerequisites exist. In this example, all members of list meet this criteria trivially.
* Place candidate growth_paths into Possible_Growth_Paths list=1.0.b

<<<< END: Discover_possible_Growth_Paths()

Step 2) Next the updater component decides on the Growth Path to pursue:

>>>> START Decide_Growth_Path()

* The growth policy dictates that we should grow to latest patched

revision. (In this example, determining the latest revision is trivial i.e. it is 1.0.b)

* chosen_growth_path=1.0.b

<<<< END: Decide_Growth_Path()

Step 3) The updater component then obtains the required resources to revise the current software level to the new one.

>>>> Get_Resources()

* Execute search on remote search engine (e.g. Internet Search Engine) using Phrase ("IBM Test REVISION 1.0.a to 1.0.b

RESOURCES").

* Search returns URL say

ftp://ftp.vendor-site/pub/test/resources/1.0.a-b"

* Updater downloads file pointed to by URL and places in secure holding area where it verifies authenticity.

* Updater verifies authenticity (using, for example, digital signatures based on RSA algorithm, or any method)

If files not authentic, then return to search (see Note 1 below)

* Updater unpacks resources into a temporary directory (see Note 2

below). These resources include machine processible installation

instructions (for example, instructions written in a script

language such as a UNIX shell script or MVS REXX) and files

(either binary or requiring compilation) which actually contain

the software fix.

<<<< END: Get_Resources()

Notes on above tasks

Note 1—To save time the updater looks for a standard file before downloading the URL called "signature", which contains the URL

ftp://ftp.vendor-site/pub/test/resources/1.0.a-b and a listing of its contents. This is hashed and signed. Using this signature, the Updater component can quickly establish authenticity of the URL (to some extent) before downloading it and use the information i.e. file listings to corroborate the final downloaded resources after they have been unpacked into the temporary directory. When the final URL is downloaded it is also checked again for authenticity (to guard against someone placing a bogus artefact in an authentic URL location).

Note 2—Part of the unpacking is that the updater component will examine the installation scripts and modify them based on the contents of its installation environment data where required. For example if the installation instructions were coded in a shell script it will replace all instances of INSTALLPATH with the token "/usr/bin/testapp/". Again Naming conventions of attributes are standardised as it the method of token substitution in installation instructions. This makes totally automatic installation possible.

Step 4) The updater component then implements the actual software upgrade:

>>>> START Install_Resources()

* execute the installation instructions.

* update the values of

Current_Installed_Version=1.0.b

Last_Growth_Time =Date+Time.

* send an e-mail to software asset manager informing of

installation and whether or not a reboot of the Operating System

or restart of the application is required before the upgrade takes affect.

<<<< END Install_Resources()

This is the end of this current growth cycle. The seed updates the Last_Growth_Time value the current time and then exits. The time taken for this cycle could be anything from a few seconds where the updater component found no upgrade paths for the currently installed version to several hours if a totally new release from the current one is to be downloaded and installed together with new pre-requisite software.

An alternative to the embodiment described above in detail does not require an independent updater component for every different software product, but uses a single generic updater component installed on a system together with product-specific plug-in objects and instructions which are downloaded with each product. These objects interoperate with the generic code to provide the same functions of the product-specific updater components described above. It will be clear to persons skilled in the art that the present invention could be implemented within systems in which some but not all application programs and other software products installed on the system have associated updater components, and that other changes to the above-described embodiments are possible within the scope of the present invention.

What is claimed is:

1. A computer program product, comprising computer program code recorded on a computer readable recording medium, the computer program code comprising an updater component for use in updating one or more computer programs installed on a computer system connected within a computer network, the updater component including:

means for initiating access to one or more identifiable locations within the network where one or more required software update resources are located, to retrieve the required software update resources;

means for performing a comparison between software update resources available from said one or more identifiable network locations and computer programs installed on said computer system, to identify available relevant update resources, and for comparing the available relevant update resources with predefined update criteria corresponding to applicable software licence terms and conditions;

means for initiating retrieval of software update resources which satisfy said predefined criteria; and

means for applying a software update to one of the installed computer programs using the one or more retrieved software resources.

2. A computer program product according to claim 1, wherein said means for applying software updates includes means for installing available relevant software resources in accordance with the predefined update criteria and in accordance with computer readable instructions for installation which are part of the software resources downloaded for the update.

3. A computer program product according to claim 1, wherein information for identifying one or more locations is

held by said updater component and includes a product identifier of a computer program product, the updater component being adapted to provide said product identifier to a search engine, the product identifier serving as a search parameter for use by said search engine to identify network locations.

4. A computer program product according to claim 3, wherein said updater component is adapted to download a list of available software update resources and their pre-requisite software products in response to said search engine identifying network locations at which said list is held, to compare the list of available software update resources and pre-requisite products with computer programs installed on said computer system and, where updates to the pre-requisite products are required, to request updates to the pre-requisite products.

5. A computer program product according to claim 1, wherein the updater component has machine readable installation instructions for installing the updater component on a computer system, the installation instructions including instructions for registering the updater component with a repository which is accessible by other updater components, such that the updater component is identifiable and contactable by other updater components.

6. A computer program product according to claim 5, wherein the updater component includes an API via which updater components of complementary computer programs can request that the current updater component update its computer program, the current updater component being adapted to call an update method to update its computer program in response to an update request, and wherein the current updater component is adapted to send a system-generated request to updater components of pre-requisite computer programs of its computer program when updating of its computer program requires updating of said pre-requisite computer programs.

7. A computer program product according to claim 12, wherein said means for applying updates is adapted to install correction and enhancement software which modifies existing installed software and also to install upgraded versions of installed software which replaces installed software.

8. A method for automated updating of a computer program installed on a computer system connected within a computer network, including the following steps:

delivering to the computer system an updater component for use in updating the computer program;

providing at a first network location downloadable software resources for building said computer program from a current version to an updated version;

wherein the updater component is adapted to perform the following steps when executed on the computer system:

(a) initiating access to said first network location at which said software resources are located;

(b) performing a comparison between software resources available from said first network location and the installed computer program, to identify available relevant update resources, and comparing the available relevant update resources with predefined update criteria corresponding to applicable software licence terms and conditions;

(c) downloading onto said computer system the available relevant software update resources which satisfy the predefined update criteria;

(d) building said computer program from the current version to the updated version using the downloaded software resources.

9. A method according to claim 19, including providing at a second network location, identifiable from information in the updater component, a computer readable list of available updates to said computer program, wherein the updater component is adapted to perform the following steps prior to accessing said first network location:

initiate access to said second network location to retrieve said list;

read said list and perform a comparison of the listed available updates with said computer program on said first computer system, thereby to identify the available relevant update resources.

* * * * *

UNITED STATES PATENT AND TRADEMARK OFFICE

CERTIFICATE OF CORRECTION

PATENT NO. : 6,199,204 B1
DATED : March 6, 2001
INVENTOR(S) : Seamus Donohue

It is certified that error appears in the above-identified patent and that said Letters Patent is hereby corrected as shown below:

Title page,
Item [75], change item [75] from "Inventor: **Seamus Donohue**, Artane (IR)" to -- Inventor: **Seamus Donohue,** Dublin, Ireland --.

Signed and Sealed this

Twenty-fourth Day of September, 2002

Attest:

JAMES E. ROGAN

Attesting Officer

Director of the United States Patent and Trademark Office

US006334212B1

(12) United States Patent
Nakajima

(10) Patent No.: US 6,334,212 B1
(45) Date of Patent: Dec. 25, 2001

(54) **COMPILER**

(75) Inventor: **Masaitsu Nakajima**, Osaka (JP)

(73) Assignee: **Matsushita Electric Industrial Co., Ltd.**, Osaka (JP)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 0 days.

(21) Appl. No.: **09/281,812**

(22) Filed: **Mar. 31, 1999**

(30) **Foreign Application Priority Data**

Apr. 1, 1998 (JP) 10-088473

(51) **Int. Cl.**[7] **G06F 9/45**
(52) **U.S. Cl.** **717/5**
(58) **Field of Search** 717/5

(56) **References Cited**

U.S. PATENT DOCUMENTS

5,148,271 *	9/1992	Kato et al.	348/390
5,440,701	8/1995	Matsuzaki et al.	712/210
5,774,737 *	6/1998	Nakano	712/24
5,790,862 *	8/1998	Tanaka et al.	395/705
5,854,850 *	12/1998	Linford et al.	382/128
5,857,103 *	1/1999	Grove	395/705
5,859,994 *	1/1999	Zaidi	712/209
5,929,869 *	7/1999	Wilde	345/508
6,009,261 *	12/1999	Scalzi et al.	395/500.47

FOREIGN PATENT DOCUMENTS

5-108373	4/1993	(JP).
7-105013	4/1995	(JP).
7-121377	5/1995	(JP).
8-314727	11/1996	(JP).

OTHER PUBLICATIONS

Suzuki et al., "An Architecture of Customized Address Space Microcomputer", Research Report of the Institute of Electronics, Information and Communication Engineers (IEICE), vol. 93 No. 320 (CPSY93-35-44), pp. 17-24.

D.R. Ditzel, et al., "The Hardware Architecture of the CRISP Microprocessor", Proceedings of the 14th Annual International Symposium on Computer Architecture, pp. 309-319, 1987.

* cited by examiner

Primary Examiner—Mark R. Powell
Assistant Examiner—John Q. Chavis
(74) *Attorney, Agent, or Firm*—McDermott, Will & Emery

(57) **ABSTRACT**

A compiler is adapted to minimize the ultimate code size of an object program that has been translated from a source program including a plurality of instructions. The compiler includes first instruction length calculator for calculating a total length of the instructions where variables for the source program are allocated to a first type of register resources in accordance with a first instruction format and second instruction length calculator for calculating a total length of the instructions where the variables are allocated to a second type of register resources in accordance with a second instruction format. The length of one instruction defined by the second instruction format is different from that defined by the first instruction format. The variables are allocated to respectively appropriate ones of the register resources based on the results of calculation derived by the first and second instruction length calculators.

10 Claims, 17 Drawing Sheets

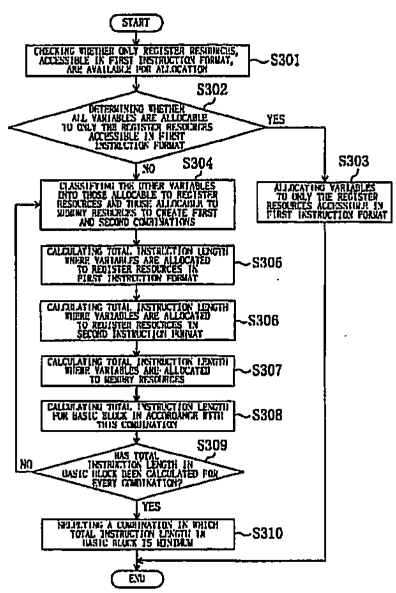


Fig. 1

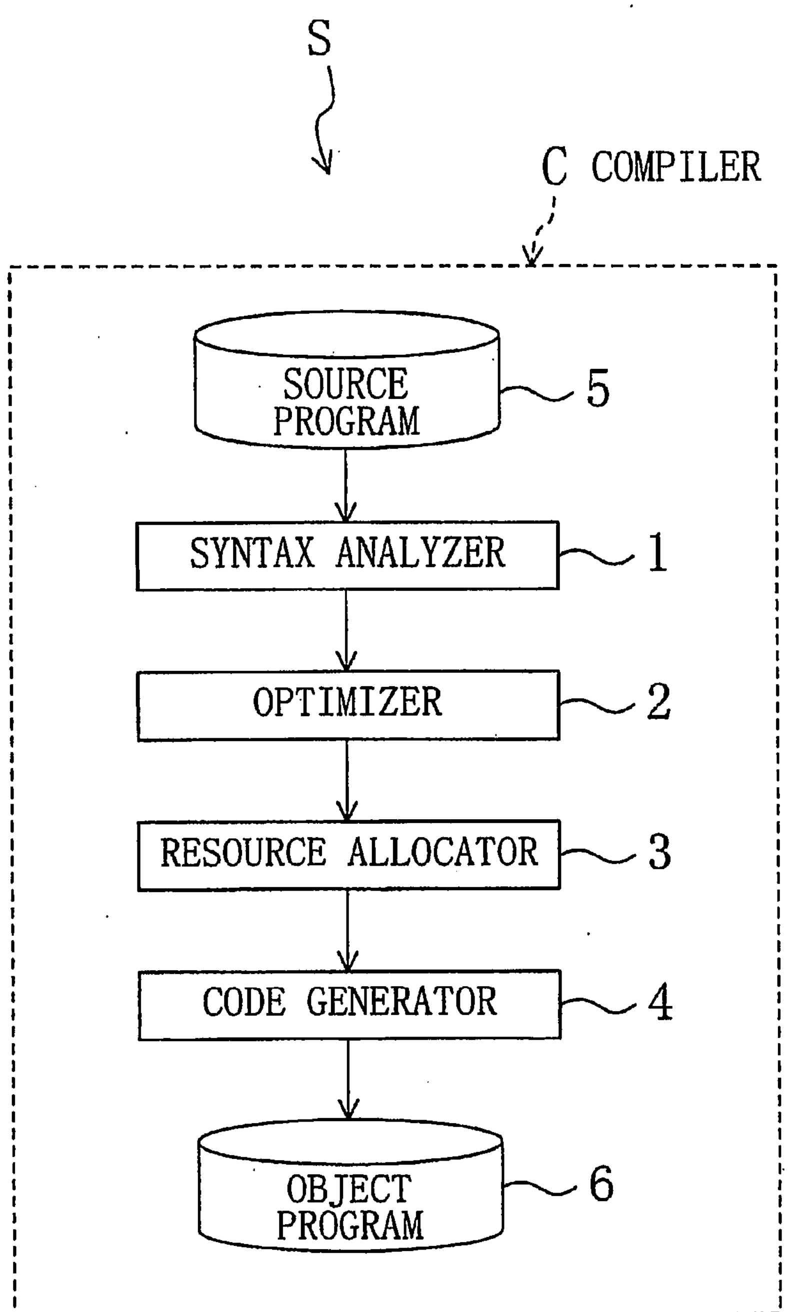

S
C COMPILER
SOURCE PROGRAM
5
SYNTAX ANALYZER
1
OPTIMIZER
2
RESOURCE ALLOCATOR
3
CODE GENERATOR
4
OBJECT PROGRAM
6

Fig. 2

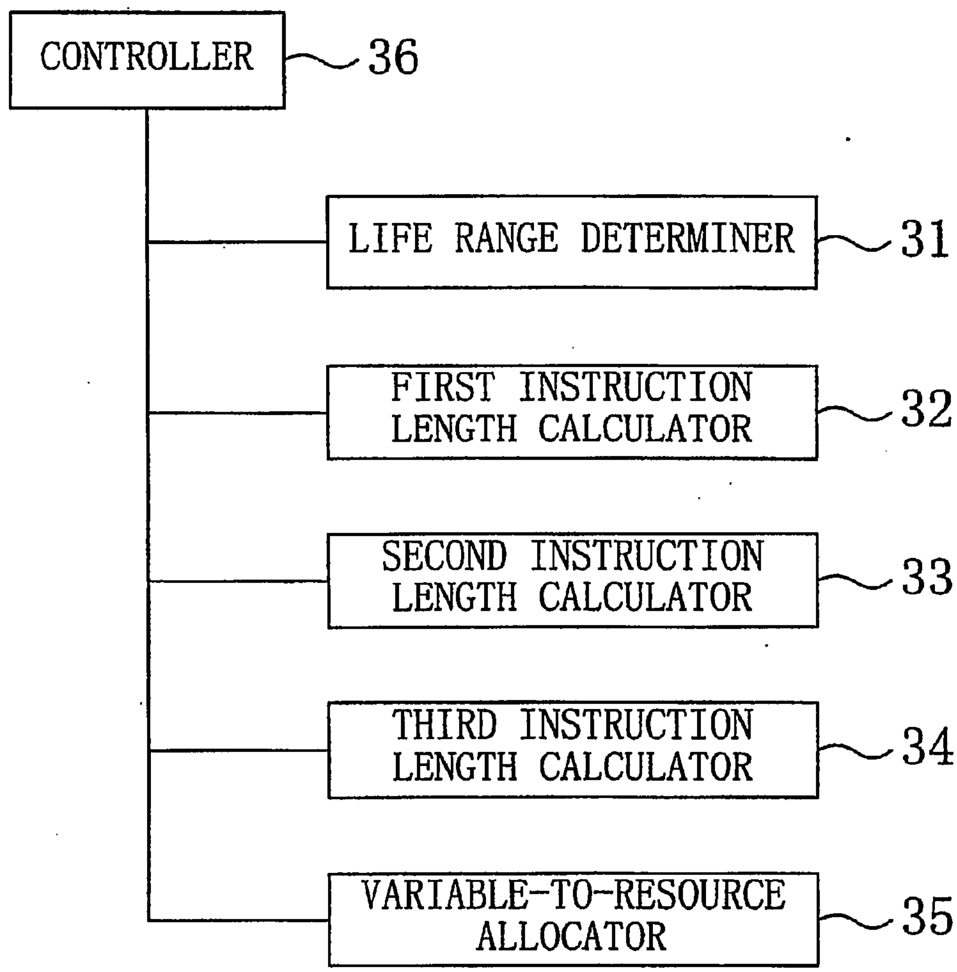

CONTROLLER
36
LIFE RANGE DETERMINER
31
FIRST INSTRUCTION LENGTH CALCULATOR
32
SECOND INSTRUCTION LENGTH CALCULATOR
33
THIRD INSTRUCTION LENGTH CALCULATOR
34
VARIABLE-TO-RESOURCE ALLOCATOR
35

Fig. 3

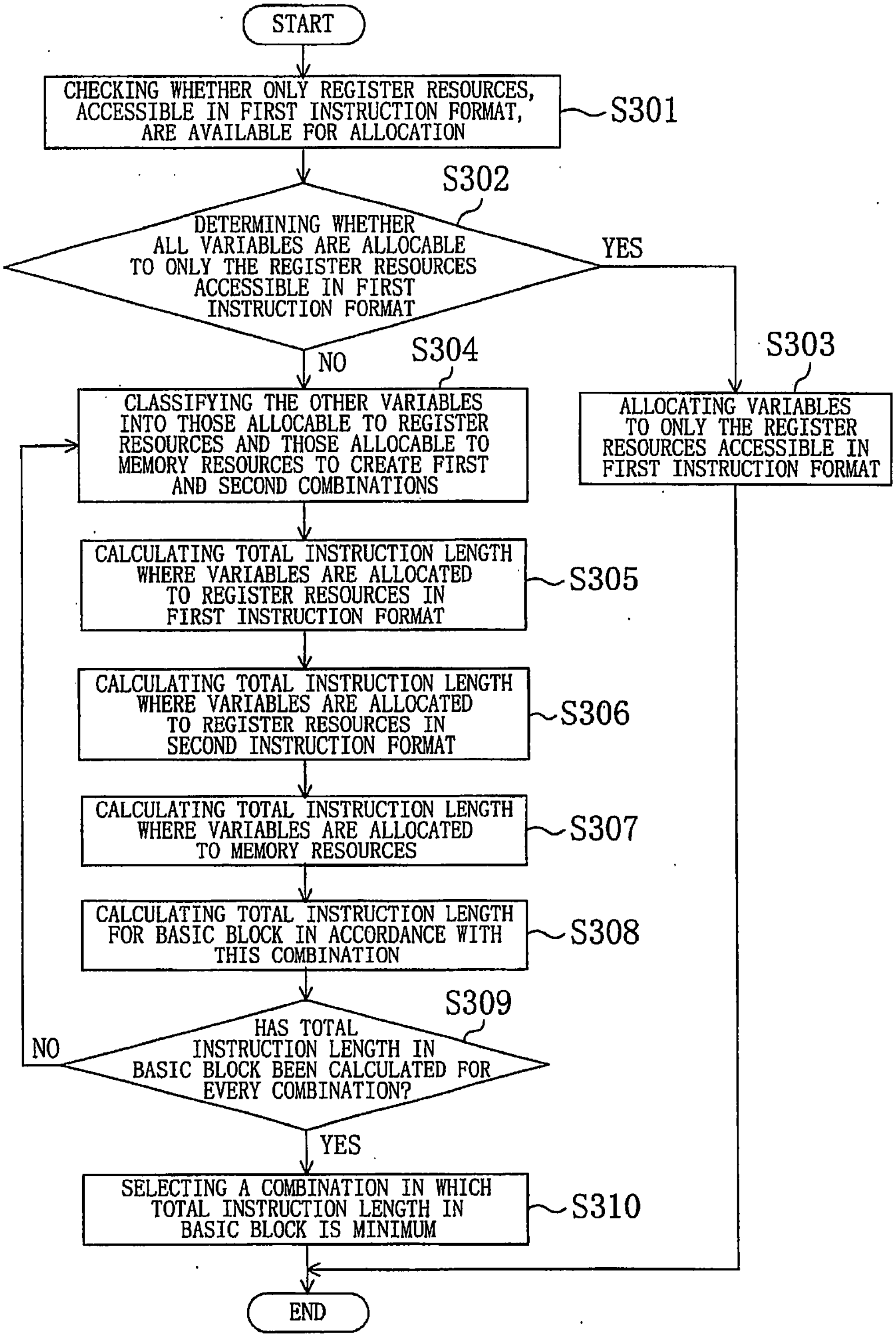

START
CHECKING WHETHER ONLY REGISTER RESOURCES, ACCESSIBLE IN FIRST INSTRUCTION FORMAT, ARE AVAILABLE FOR ALLOCATION
S301
S302
DETERMINING WHETHER ALL VARIABLES ARE ALLOCABLE TO ONLY THE REGISTER RESOURCES ACCESSIBLE IN FIRST INSTRUCTION FORMAT
YES
NO
S303
ALLOCATING VARIABLES TO ONLY THE REGISTER RESOURCES ACCESSIBLE IN FIRST INSTRUCTION FORMAT
S304
CLASSIFYING THE OTHER VARIABLES INTO THOSE ALLOCABLE TO REGISTER RESOURCES AND THOSE ALLOCABLE TO MEMORY RESOURCES TO CREATE FIRST AND SECOND COMBINATIONS
CALCULATING TOTAL INSTRUCTION LENGTH WHERE VARIABLES ARE ALLOCATED TO REGISTER RESOURCES IN FIRST INSTRUCTION FORMAT
S305
CALCULATING TOTAL INSTRUCTION LENGTH WHERE VARIABLES ARE ALLOCATED TO REGISTER RESOURCES IN SECOND INSTRUCTION FORMAT
S306
CALCULATING TOTAL INSTRUCTION LENGTH WHERE VARIABLES ARE ALLOCATED TO MEMORY RESOURCES
S307
CALCULATING TOTAL INSTRUCTION LENGTH FOR BASIC BLOCK IN ACCORDANCE WITH THIS COMBINATION
S308
S309
HAS TOTAL INSTRUCTION LENGTH IN BASIC BLOCK BEEN CALCULATED FOR EVERY COMBINATION?
NO
YES
SELECTING A COMBINATION IN WHICH TOTAL INSTRUCTION LENGTH IN BASIC BLOCK IS MINIMUM
S310
END

Fig. 4

INTERMEDIATE LANGUAGE PROGRAM

```
s1:  t1=p1+p2
s2:  t2=p1-p2
s3:  t3=p1*p2
s4:  t4=t1+p1
s5:  t5=t2*p1
s6:  t6=t4+p2
s7:  t7=t3*p1
s8:  t8=t4+5
```

BASIC BLOCK

Fig. 5

```
s1:  t1=p1+p2
s2:  t2=p1-p2
s3:  t3=p1*p2
s4:  t4=t1+p1
s5:  t5=t2*p1
s6:  t6=t4+p2
s7:  t7=t3*p1
s8:  t8=t4+5
```

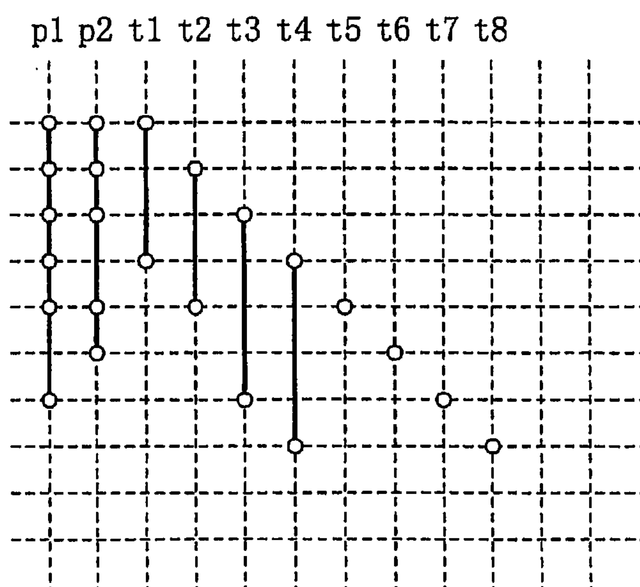


Fig. 6

ALLOCATION OF VARIABLES TO RESOURCES

p1	E0
p2	E1
t1	E2
t2	E3
t3	E4
t4	E5
t5	E6
t6	E7
t7	D2
t8	D3

INTERMEDIATE LANGUAGE PROGRAM

```
s1:  t1=p1+p2
s2:  t2=p1-p2
s3:  t3=p1*p2
s4:  t4=t1+p1
s5:  t5=t2*p1
s6:  t6=t4+p2
s7:  t7=t3*p1
s8:  t8=t4+5
```

MACHINE INSTRUCTION PROGRAM

```
s1:  mov   E0,E2   3
     add   E1,E2   3

s2:  mov   E0,E3   3
     sub   E1,E3   3

s3:  mov   E0,E4   3
     mul   E1,E4   3

s4:  mov   E0,E5   3
     add   E2,E5   3

s5:  mov   E0,E6   3
     mul   E3,E6   3

s6:  mov   E1,E7   3
     add   E5,E7   3

s7:  mov   E0,D2   2
     mul   E4,D2   3

s8:  mov   E3,D3   2
     add    5,D3   3
                 ---------
                  46byte
```

Fig. 7

ALLOCATION OF VARIABLES TO RESOURCES

p1	D2
p2	D3
t1	E0
t2	E1
t3	E2
t4	E3
t5	E4
t6	E5
t7	E6
t8	E7

INTERMEDIATE LANGUAGE PROGRAM

```
s1:  t1=p1+p2
s2:  t2=p1-p2
s3:  t3=p1*p2
s4:  t4=t1+p1
s5:  t5=t2*p1
s6:  t6=t4+p2
s7:  t7=t3*p1
s8:  t8=t4+5
```

MACHINE INSTRUCTION PROGRAM

```
s1:  mov   D2, E0   2
     add   D3, E0   3

s2:  mov   D2, E1   2
     sub   D3, E1   3

s3:  mov   D2, E2   2
     mul   D3, E2   3

s4:  mov   D2, E3   2
     add   E0, D3   3

s5:  mov   D2, E4   2
     mul   E1, E4   3

s6:  mov   D3, E5   2
     add   E3, E5   3

s7:  mov   D2, E6   2
     mul   E2, E6   3

s8:  mov   E3, E7   3
     add    5, E7   4
                 ---------
                  42byte
```

Fig. 8

ALLOCATION OF VARIABLES TO RESOURCES

p1	D2
p2	D3
t1	E0
t2	E1
t3	E2
t4	E3
t5	E4
t6	E5
t7	E6
t8	E7

INTERMEDIATE LANGUAGE PROGRAM

```
s1:  t1=p1+p2
s2:  t2=p1-p2
s3:  t3=p1*p2
s4:  t4=t1+p1
s5:  t5=t2*p1
s6:  t6=t4+p2
s7:  t7=t3*p1
s8:  t8=t4+5
```

MACHINE INSTRUCTION PROGRAM

```
s1:  mov   D2,D0   1
     add   D3,D0   1
     mov   D0,E0   2

s2:  mov   D2,D0   1
     sub   D3,D0   2
     mov   D0,E1   2

s3:  mov   D2,D0   1
     mul   D3,D0   2
     mov   D0,E2   2

s4:  mov   D2,D0   1
     add   E0,D0   3
     mov   D0,E3   2

s5:  mov   D2,D0   1
     mul   E1,D0   3
     mov   D0,E4   2

s6:  mov   D3,D0   1
     add   E3,D0   3
     mov   D0,E5   2

s7:  mov   D2,D0   1
     mul   E2,D0   3
     mov   D0,E6   2

s8:  mov   E3,D0   2
     add    5,D0   2
     mov   D0,E7   2
                 ---------
                  44byte
```

Fig. 9

ALLOCATION OF VARIABLES TO RESOURCES

p1	D2
p2	D3
t1	E0
t2	E1
t3	E2
t4	E3
t5	E4
t6	E5
t7	E6
t8	E7

INTERMEDIATE LANGUAGE PROGRAM

```
s1:  t1=p1+p2
s2:  t2=p1-p2
s3:  t3=p1*p2
s4:  t4=t1+p1
s5:  t5=t2*p1
s6:  t6=t4+p2
s7:  t7=t3*p1
s8:  t8=t4+5
```

MACHINE INSTRUCTION PROGRAM

```
s1:  mov   D2,D0   1
     add   D3,D0   1
     mov   D0,E0   2

s2:  mov   D2,E1   2
     sub   D3,E1   3

s3:  mov   D2,E2   2
     mul   D3,E2   3

s4:  mov   D2,E3   2
     add   E0,E3   3

s5:  mov   D2,E4   2
     mul   E1,E4   3

s6:  mov   D3,E5   2
     add   E3,E5   3

s7:  mov   D2,E6   2
     mul   E2,E6   3

s8:  mov   E3,D0   2
     add    5,D0   2
     mov   D0,E7   2
                ---------
                 40byte
```

Fig. 10

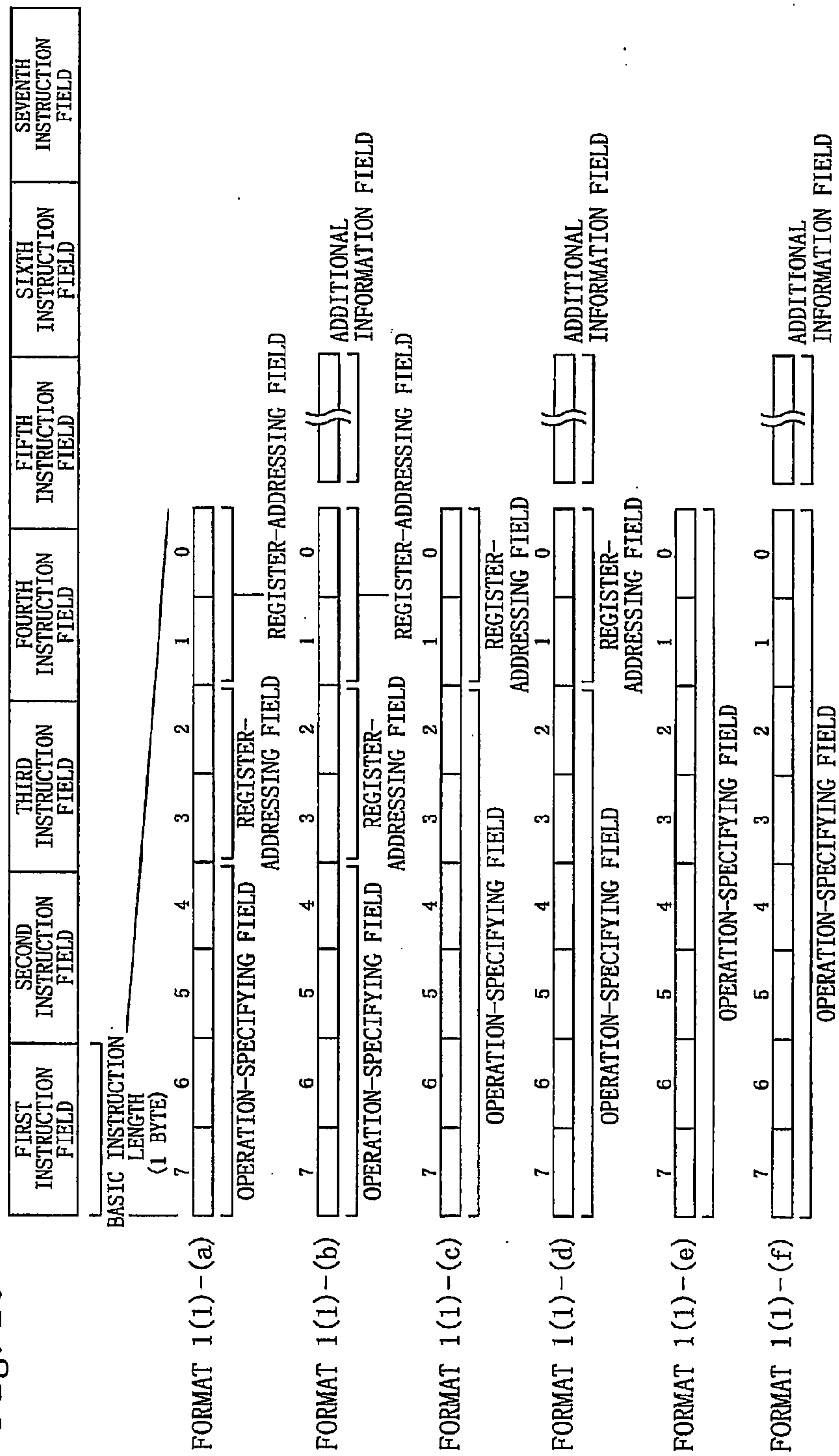

FIRST INSTRUCTION FORMAT (1)
FIRST INSTRUCTION FIELD
SECOND INSTRUCTION FIELD
THIRD INSTRUCTION FIELD
FOURTH INSTRUCTION FIELD
FIFTH INSTRUCTION FIELD
SIXTH INSTRUCTION FIELD
SEVENTH INSTRUCTION FIELD
BASIC INSTRUCTION LENGTH (1 BYTE)
FORMAT 1(1)-(a)
FORMAT 1(1)-(b)
FORMAT 1(1)-(c)
FORMAT 1(1)-(d)
FORMAT 1(1)-(e)
FORMAT 1(1)-(f)
7 6 5 4 3 2 1 0
OPERATION-SPECIFYING FIELD
REGISTER-ADDRESSING FIELD
ADDITIONAL INFORMATION FIELD

Fig. 11

```
FIRST INSTRUCTION FORMAT (1)-(a)
  ADD Dm, Dn          : ADD
  CMP Dm, Dn          : COMPARE
  CMP Am, An          : COMPARE
  MOV (Am), Dn        : TRANSFER FROM MEMORY TO REGISTER (LOAD)
  MOV Dm, (An)        : TRANSFER FROM REGISTER TO MEMORY (STORE)
  MOV Dm, Dn          : TRANSFER FROM REGISTER TO REGISTER
  MOV Am, An          : TRANSFER FROM REGISTER TO REGISTER
  ...

FIRST INSTRUCTION FORMAT (1)-(b)
  ...

FIRST INSTRUCTION FORMAT (1)-(c)
  CLR Dm              : CLEAR DATA TO 0
  INC Dm              : INCREASE DATA BY 1
  INC Am              : INCREASE DATA BY 1
  EXTB Dm             : EXTEND SIGNED BYTE DATA INTO WORD DATA
  EXTUB Dm            : EXTEND UNSIGNED BYTE DATA INTO WORD DATA
  EXTH Dm             : EXTEND SIGNED HALF-WORD DATA INTO WORD DATA
  EXTHU Dm            : EXTEND UNSIGNED HALF-WORD DATA INTO WORD DATA
  MOV SP, An          : TRANSFER STACK POINTER (SP) TO REGISTER
  ...

FIRST INSTRUCTION FORMAT (1)-(d)
  ADD imm8, An        : ADD 8-BIT IMMEDIATE VALUE
  ADD imm8, Dn        : ADD 8-BIT IMMEDIATE VALUE
  MOV imm16, An       : TRANSFER 16-BIT IMMEDIATE VALUE
  MOV imm16, Dn       : TRANSFER 16-BIT IMMEDIATE VALUE
  MOV (abs16), Dn     : TRANSFER FROM MEMORY TO REGISTER (LOAD) BY ADDRESSING WITH 16-BIT ABSOLUTE VALUE
  MOVBU (abs16), Dn   : TRANSFER ZERO-EXTENDED BYTE DATA FROM MEMORY TO REGISTER (LOAD)
                        BY ADDRESSING WITH 16-BIT ABSOLUTE VALUE
  MOVHU (abs16), Dn   : TRANSFER ZERO-EXTENDED HALF-WORD DATA FROM MEMORY TO REGISTER (LOAD)
                        BY ADDRESSING WITH 16-BIT ABSOLUTE VALUE
  MOV Dm, (abs16)     : TRANSFER FROM REGISTER TO MEMORY (STORE) BY ADDRESSING WITH 16-BIT ABSOLUTE VALUE
  MOVBU Dm, (abs16)   : TRANSFER ZERO-EXTENDED BYTE DATA FROM REGISTER TO MEMORY (STORE)
                        BY ADDRESSING WITH 16-BIT ABSOLUTE VALUE
  MOVHU Dm(abs16)     : TRANSFER ZERO-EXTENDED HALF-WORD DATA FROM REGISTER TO MEMORY (STORE)
                        BY ADDRESSING WITH 16-BIT ABSOLUTE VALUE
  MOV (disp8, SP), An : TRANSFER FROM MEMORY TO REGISTER (LOAD) BY ADDRESSING USING STACK POINTER (SP) WITH DISPLACEMENT
  MOV (disp8, SP), Dn : TRANSFER FROM MEMORY TO REGISTER (LOAD) BY ADDRESSING USING STACK POINTER (SP) WITH DISPLACEMENT
  MOV Am, (disp8, SP) : TRANSFER FROM REGISTER TO MEMORY (STORE) BY ADDRESSING USING STACK POINTER (SP) WITH DISPLACEMENT
  MOV Dm, (disp8, SP) : TRANSFER FROM REGISTER TO MEMORY (STORE) BY ADDRESSING USING STACK POINTER (SP) WITH DISPLACEMENT
  ...

FIRST INSTRUCTION FORMAT (1)-(e)
  NOP                 : NO OPERATION
  LOOP #CC            : LOOP CONTROL INSTRUCTION
  ...

FIRST INSTRUCTION FORMAT (1)-(f)
  BR #CC(disp8, PC)   : CONDITIONAL BRANCH OF PC WITH DISPLACEMENT
  JMP (disp8, PC)     : UNCONDITIONAL BRANCH OF PC WITH DISPLACEMENT
  ...
```

Fig. 12

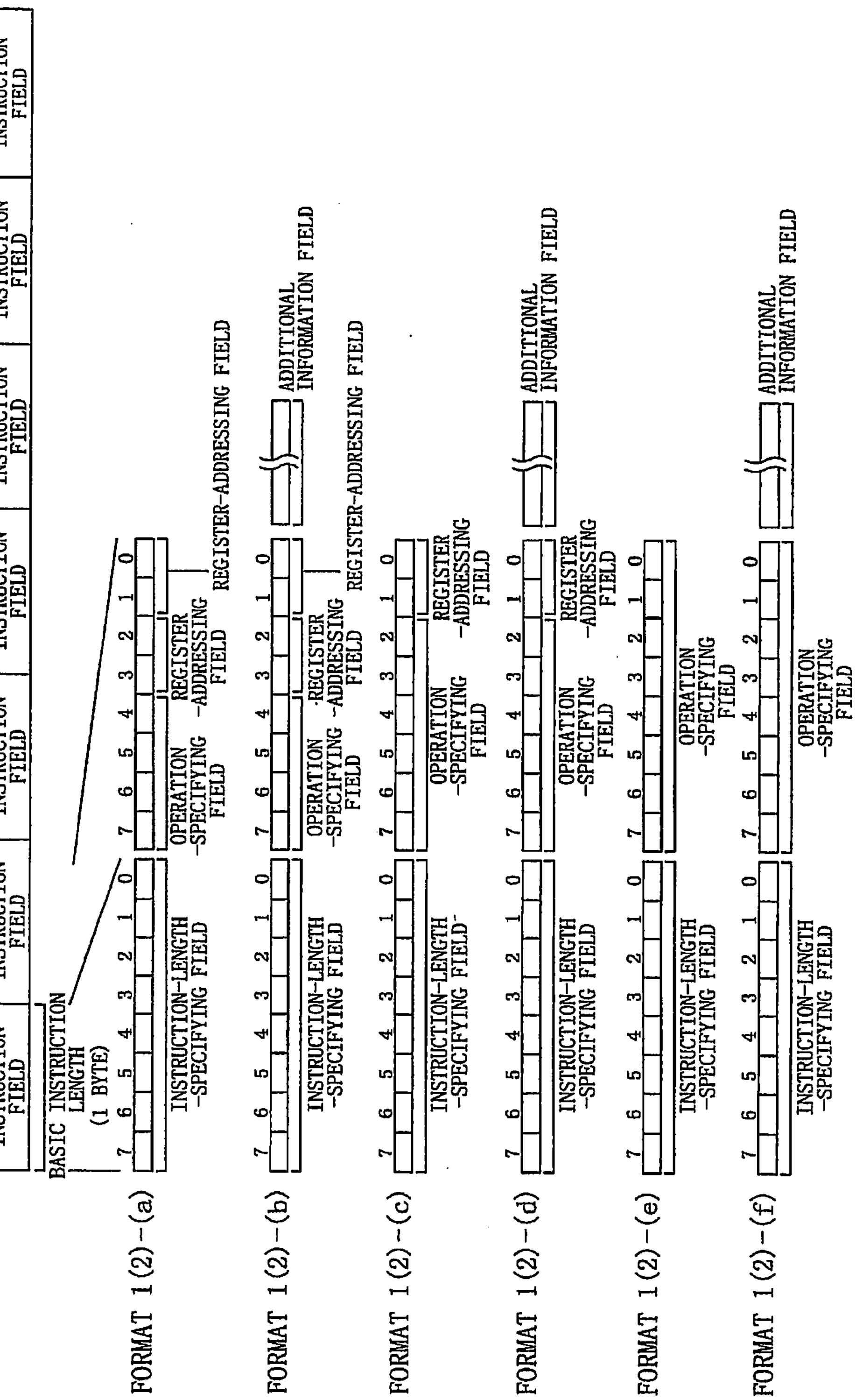

FIRST INSTRUCTION FORMAT (2)
FIRST INSTRUCTION FIELD
SECOND INSTRUCTION FIELD
THIRD INSTRUCTION FIELD
FOURTH INSTRUCTION FIELD
FIFTH INSTRUCTION FIELD
SIXTH INSTRUCTION FIELD
SEVENTH INSTRUCTION FIELD
BASIC INSTRUCTION LENGTH (1 BYTE)
FORMAT 1(2)-(a)
FORMAT 1(2)-(b)
FORMAT 1(2)-(c)
FORMAT 1(2)-(d)
FORMAT 1(2)-(e)
FORMAT 1(2)-(f)
7 6 5 4 3 2 1 0
INSTRUCTION-LENGTH -SPECIFYING FIELD
OPERATION -SPECIFYING FIELD
REGISTER -ADDRESSING FIELD
REGISTER-ADDRESSING FIELD
ADDITIONAL INFORMATION FIELD

Fig. 13

```
FIRST INSTRUCTION FORMAT (2)-(a)
  SUB Dm,Dn          : SUBTRACT
  MOV(Am),An         : TRANSFER FROM MEMORY TO REGISTER (LOAD)
  MOV Am,(An)        : TRANSFER FROM REGISTER TO MEMORY (STORE)
  ...

FIRST INSTRUCTION FORMAT (2)-(b)
  MOV(Ai,Dn),Dn      : TRANSFER FROM MEMORY TO REGISTER (LOAD)
  ...                  INDIRECTLY BY WAY OF INDEXED REGISTER

FIRST INSTRUCTION FORMAT (2)-(c)
  ...

FIRST INSTRUCTION FORMAT (2)-(d)
  ADD imm16,An       : ADD 16-BIT IMMEDIATE VALUE
  ADD imm16,Dn       : ADD 16-BIT IMMEDIATE VALUE
  ...

FIRST INSTRUCTION FORMAT (2)-(e)
  RTI                : RETURN FROM INTERRUPT STATE
  ...

FIRST INSTRUCTION FORMAT (2)-(f)
  ...
```

Fig. 14

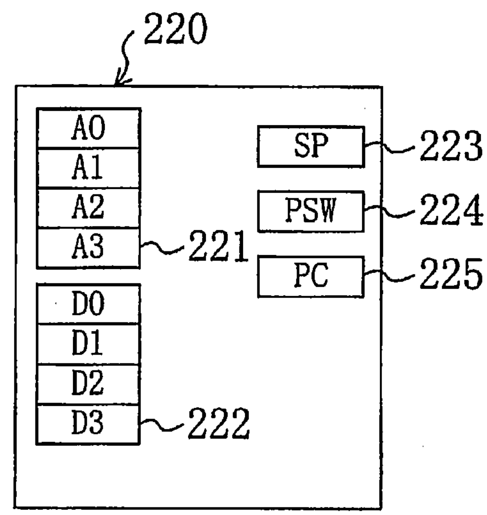


Fig. 15

NAME OF REGISTER	BIT ASSIGNMENT ON INSTRUCTION CODE	NUMBER OF PHYSICAL REGISTER	NAME OF PHYSICAL REGISTER
A0	00	00	ADDRESS REGISTER
A1	01	01	ADDRESS REGISTER
A2	02	02	ADDRESS REGISTER
A3	03	03	ADDRESS REGISTER
D0	00	00	ADDRESS REGISTER
D1	01	01	ADDRESS REGISTER
D2	02	02	ADDRESS REGISTER
D3	03	03	ADDRESS REGISTER

Fig. 16

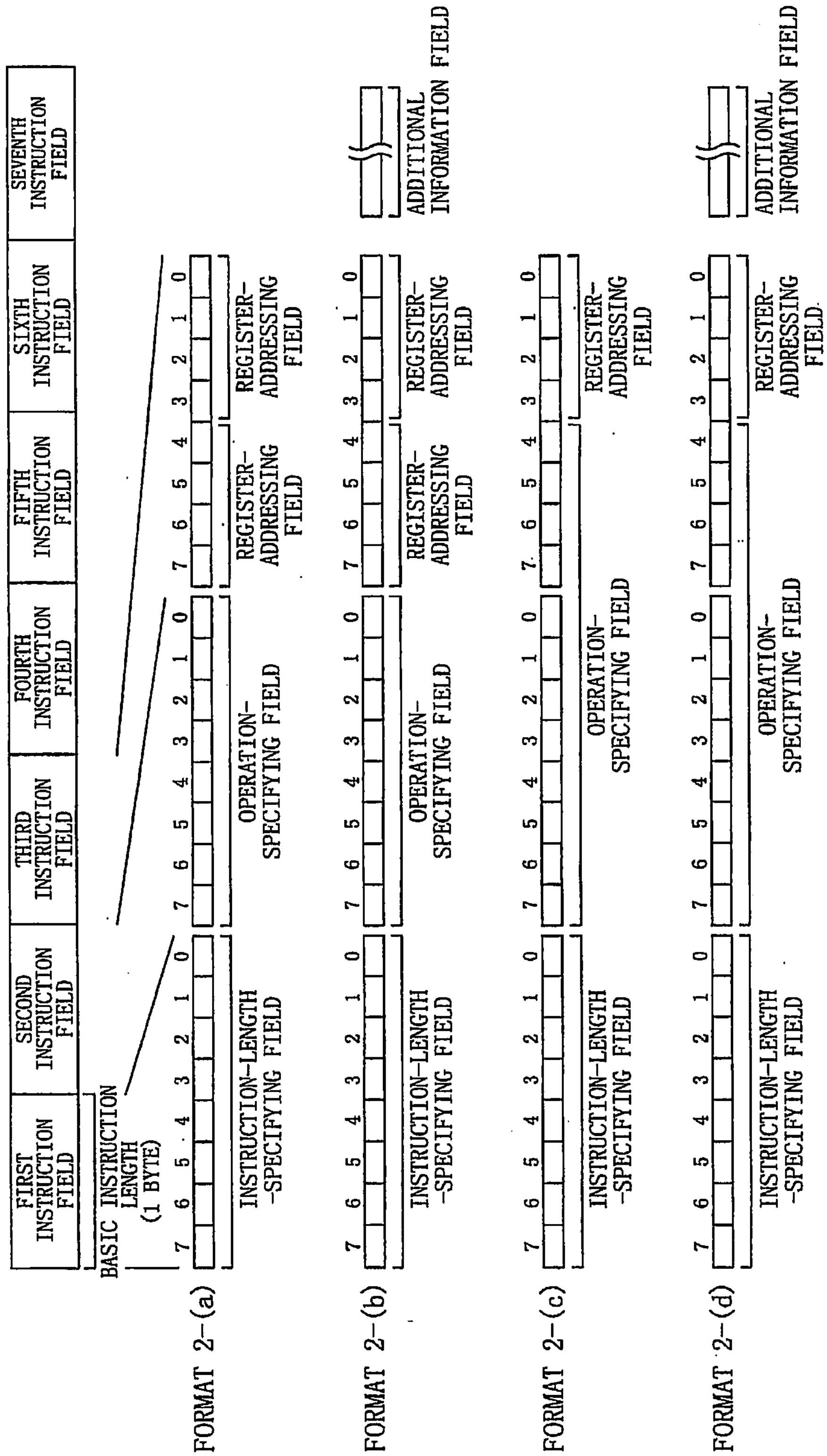

SECOND INSTRUCTION FORMAT
FIRST INSTRUCTION FIELD
SECOND INSTRUCTION FIELD
THIRD INSTRUCTION FIELD
FOURTH INSTRUCTION FIELD
FIFTH INSTRUCTION FIELD
SIXTH INSTRUCTION FIELD
SEVENTH INSTRUCTION FIELD
BASIC INSTRUCTION LENGTH (1 BYTE)
FORMAT 2-(a)
FORMAT 2-(b)
FORMAT 2-(c)
FORMAT 2-(d)
7 6 5 4 3 2 1 0
INSTRUCTION-LENGTH -SPECIFYING FIELD
OPERATION-SPECIFYING FIELD
REGISTER-ADDRESSING FIELD
ADDITIONAL INFORMATION FIELD

Fig. 17

```
SECOND INSTRUCTION FORMAT (a)
  ADD Rm, Rn          : ADD
  SUB Rm, Rn          : SUBTRACT
  CMP Rm, Rn          : COMPARE
  MOV(Rm), Rn         : TRANSFER FROM MEMORY TO REGISTER (LOAD)
  MOV Rm, (Rn)        : TRANSFER FROM REGISTER TO MEMORY (STORE)
  MOV Rm, Rn          : TRANSFER FROM REGISTER TO REGISTER
  ...

SECOND INSTRUCTION FORMAT (b)
  ADD Rm, Rn, Rd      : ADD
  SUB Rm, Rn, Rd      : SUBTRACT
  MOV(Ri, Rm), Rn     : TRANSFER FROM MEMORY TO REGISTER (LOAD)
  ...                   INDIRECTLY BY WAY OF INDEXED REGISTER

SECOND INSTRUCTION FORMAT (c)
  ...

SECOND INSTRUCTION FORMAT (d)
  ADD imm 16, Rn      : ADD 16-BIT IMMEDIATE VALUE
  ADD imm 16, Rn      : ADD 16-BIT IMMEDIATE VALUE
  MOV(disp8, SP), Rn  : TRANSFER FROM MEMORY TO REGISTER (LOAD)
                        BY ADDRESSING USING STACK POINTER (SP) WITH DISPLACEMENT
  MOV Rm, (disp8, SP) : TRANSFER FROM REGISTER TO MEMORY (STORE)
  ...                   BY ADDRESSING USING STACK POINTER (SP) WITH DISPLACEMENT
```

Fig. 18

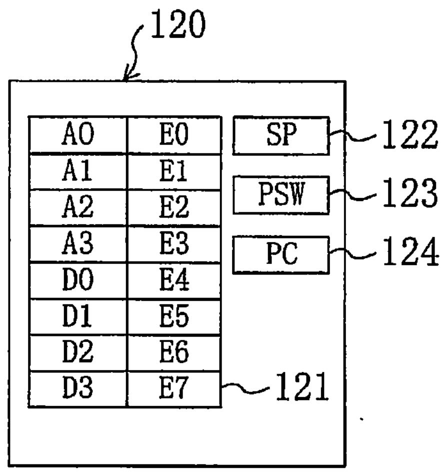


Fig. 19

NAME OF REGISTER	BIT ASSIGNMENT ON INSTRUCTION CODE	NUMBER OF PHYSICAL REGISTER	NAME OF PHYSICAL REGISTER
A0	00	1000	GENERAL-PURPOSE REGISTER
A1	01	1001	GENERAL-PURPOSE REGISTER
A2	10	1010	GENERAL-PURPOSE REGISTER
A3	11	1011	GENERAL-PURPOSE REGISTER
D0	00	1100	GENERAL-PURPOSE REGISTER
D1	01	1101	GENERAL-PURPOSE REGISTER
D2	10	1110	GENERAL-PURPOSE REGISTER
D3	11	1111	GENERAL-PURPOSE REGISTER
E0			
E1			
E2			
E3			
E4			
E5			
E6			
E7			

Fig. 20

NAME OF REGISTER	BIT ASSIGNMENT ON INSTRUCTION CODE	NUMBER OF PHYSICAL REGISTER	NAME OF PHYSICAL REGISTER
A0	1000	1000	GENERAL-PURPOSE REGISTER
A1	1001	1001	GENERAL-PURPOSE REGISTER
A2	1010	1010	GENERAL-PURPOSE REGISTER
A3	1011	1011	GENERAL-PURPOSE REGISTER
D0	1100	1100	GENERAL-PURPOSE REGISTER
D1	1101	1101	GENERAL-PURPOSE REGISTER
D2	1110	1110	GENERAL-PURPOSE REGISTER
D3	1111	1111	GENERAL-PURPOSE REGISTER
E0	0000	0000	GENERAL-PURPOSE REGISTER
E1	0001	0001	GENERAL-PURPOSE REGISTER
E2	0010	0010	GENERAL-PURPOSE REGISTER
E3	0011	0011	GENERAL-PURPOSE REGISTER
E4	0100	0100	GENERAL-PURPOSE REGISTER
E5	0101	0101	GENERAL-PURPOSE REGISTER
E6	0110	0110	GENERAL-PURPOSE REGISTER
E7	0111	0111	GENERAL-PURPOSE REGISTER

COMPILER

BACKGROUND OF THE INVENTION

The present invention relates to a compiler for translating a source program written in a high-level programming language into an object program written in a machine language.

In recent years, programmers have been trying very hard to improve the efficiency in developing a program by writing a program in a high-level programming language like C. The use of a high-level programming language enables a programmer to arbitrarily define a desired number of steps of holding, computing or transferring numerical values in a program using variables. That is to say, a programmer can freely write a program. During this process, a program written in such a high-level programming language (i.e., source program, which is also often called a "source code file") should be compiled, or translated, by a compiler into an object program written in a computer-executable machine language (which is often called an "object code file"). The steps in the machine-executable object program are represented by machine instructions, which require registers or memories as operands. Accordingly, variables should be allocated to these registers or memories. Such allocation processing is called "resource allocation". If optimum resource allocation has been performed successfully, then the code size of the object program can be minimized.

In general, allocating respective variables to registers turns out to be more advantageous in terms of code size and execution time rather than allocating them to memories. However, generally speaking, the number of available registers is relatively small. Thus, the degree of optimization achievable in the resource allocation solely depends on how efficiently variables can be allocated to register resources to execute a machine instruction using the registers as operands. In accordance with a conventional technique of optimizing resources allocation, a plurality of variables, allocable to the same register, are identified based on the respective ranges where the stored values of these variables are alive (in this specification, such a range will be called "variable life range"). Based on the results of this identification, the variables are allocated to the resources.

The present inventors proposed a data processor using the following two types of instruction formats and register models for the execution of instructions in Japanese Patent Application No. 10-59680.

FIGS. **10** through **20** outline the first instruction format.

In the first instruction format, a variable-length instruction with a minimum instruction length of 1 byte is described. A 2-bit field is used as a register-addressing field. Accordingly, four registers can be specified with one register-addressing field. In this architecture, four address registers and four data registers are defined. By separately using the address registers or the data registers responsive to a specific instruction, eight registers can be used in total in executing an instruction.

FIG. **10** illustrates a bit assignment for the first instruction format (1) in which a first instruction field composed of 1 byte, equal to the minimum instruction length, consists of an operation-specifying field and an arbitrary number of register-addressing fields. Specific examples of this format will be described below.

In an exemplary first instruction format (1)-(a), the first instruction field includes two 2-bit register-addressing fields and is composed of 1 byte, which is the minimum instruction length. And two operands can be specified in accordance with this format.

In another exemplary first instruction format (1)-(b), the first instruction field includes two 2-bit register-addressing fields, and an additional information field is further provided. Thus, the instruction length in accordance with this format is 2 bytes or more in total.

In still another exemplary first instruction format (1)-(c), the first instruction field includes one 2-bit register-addressing field and is composed of 1 byte, which is the minimum instruction length. And one operand can be specified in accordance with this format.

In yet another exemplary first instruction format (1)-(d), the first instruction field includes one 2-bit register-addressing field, and an additional information field is further provided. Thus, the instruction length in accordance with this format is 2 bytes or more in total.

In yet another exemplary first instruction format (1)-(e), the first instruction field includes no register-addressing fields and is composed of 1 byte, which is the minimum instruction length. Accordingly, in accordance with this format, no operands can be specified using addresses.

In yet another exemplary first instruction format (1)-(f), the first instruction field includes no register-addressing fields but an additional information field is further provided. Thus, the instruction length in accordance with this format is 2 bytes or more in total.

FIG. **11** illustrates part of a list of specific instructions for respective types of bit assignment shown in FIG. **10**. In FIG. **11**, instruction mnemonics are shown on the left and the operations performed to execute these instructions are shown on the right.

FIG. **12** illustrates a bit assignment for a first instruction format (2) in which a first instruction field composed of 1 byte, i.e., the minimum instruction length, consists of an instruction-length-specifying field and a second instruction field consists of an operation-specifying field and an arbitrary number of register-addressing fields. Specific examples of this format will be described in detail below.

In an exemplary first instruction format (2)-(a), the second instruction field includes two 2-bit register-addressing fields and the first and second instruction fields are composed of 2 bytes. And two operands can be specified in accordance with this format.

In another exemplary first instruction format (2)-(b), the second instruction field includes two 2-bit register-addressing fields, and an additional information field is further provided. Thus, the instruction length in accordance with this format is 3 bytes or more in total.

In still another exemplary first instruction format (2)-(c), the second instruction field includes one 2-bit register-addressing field and the first and second instruction fields are composed of 2 bytes. And one operand can be specified in accordance with this format.

In yet another exemplary first instruction format (2)-(d), the second instruction field includes one 2-bit register-addressing field, and an additional information field is further provided. Thus, the instruction length in accordance with this format is 3 bytes or more in total.

In yet another exemplary first instruction format (2)-(e), the second instruction field includes no register-addressing fields and the first and second instruction fields are composed of 2 bytes. Accordingly, in accordance with this format, no operands can be specified using addresses.

In yet another exemplary first instruction format (2)-(f), the second instruction field includes no register-addressing

fields but an additional information field is further provided. Thus, the instruction length in accordance with this format is 3 bytes or more in total.

FIG. 13 illustrates part of a list of specific instructions for respective types of bit assignment shown in FIG. 12. In FIG. 13, instruction mnemonics are shown on the left and the operations performed to execute these instructions are shown on the right.

Accordingly, in accordance with the first instruction format shown in FIGS. 10 through 13, the instruction length of the first instruction field is used as a basic instruction length to specify a variable-length instruction. And an instruction can be described in this format to have a length N times as large as the basic instruction length and equal to or less than the maximum instruction length, which is M times as large as the basic instruction length (where N and M are both positive integers and $1 \leqq N \leqq M$). Since the minimum instruction length is 1 byte, this instruction format is suitable for downsizing a program.

FIG. 14 illustrates a first register file **220** included in the data processor proposed by the present inventors. The first register file **220** includes: four address registers **A0** through **A3**; four data registers **D0** through **D3**; a stack pointer (SP) **223**; a processor status word (PSW) **224** for holding internal status information and control information; and a program counter (PC) **225**.

FIG. **15** is a table illustrating accessing the address and data registers **A0** through **A3** and **D0** through **D3** included in the first register file **220** in greater detail. Specifically, this is a table of correspondence among name of a register specified by an instruction, bit assignment on an instruction code specified in a register-addressing field, and number and name of a physical register to be accessed.

In the first instruction format, the set of instruction addressing fields specified by respective instructions to access the four address registers **A0** through **A3** is the same as the set of instruction addressing fields specified by respective instructions to access the four data registers **D0** through **D3** as shown in FIG. **15**. That is to say, the same 2-bit instruction addressing field is used to address a desired register, and it is determined by the operation of the instruction itself whether an address register or a data register should be accessed.

Next, respective bit assignments for a second instruction format, which is added as an extension to the first instruction format shown in FIGS. **10** and **12**, i.e., the basic instruction format of this architecture, will be described with reference to FIG. **16**.

In each of the bit assignments shown in FIG. **16** for the second instruction format, a first instruction field, composed of 1 byte, which is the minimum instruction length, consists of an instruction-length-specifying field. And second and third instruction fields consist of an operation-specifying field and an arbitrary number of register-addressing fields. In accordance with the second instruction format, each register-addressing field is composed of 4 bits. Specific examples of this format will be described in detail below.

In an exemplary second instruction format (a), the third instruction field includes two 4-bit register-addressing fields and the first through third instruction fields are composed of 3 bytes in total. And two operands can be specified in accordance with this format.

In another exemplary second instruction format (b), the third instruction field also includes two 4-bit register-addressing fields, and an additional information field is further provided. Thus, the instruction length in accordance with this format is 4 bytes or more in total.

In still another exemplary second instruction format (c), the third instruction field includes one 4-bit register-addressing field and the first through third instruction fields are composed of 3 bytes in total. And one operand can be specified in accordance with this format.

In yet another exemplary second instruction format (d), the third instruction field includes one 4-bit register-addressing field, and an additional information field is further provided. Thus, the instruction length in accordance with this format is 4 bytes or more in total.

Thus, in accordance with the second instruction format, the instruction length of the first instruction field is also used as a basic instruction length. And an instruction can be described in this format to have a variable length N times as large as the basic instruction length and equal to or less than the maximum instruction length, which is M times as large as the basic instruction length (where N and M are both positive integers and $1 \leqq N \leqq M$).

FIG. **17** illustrates part of a list of specific instructions for respective types of bit assignment shown in FIG. **16**. In FIG. **17**, instruction mnemonics are shown on the left and the operations performed to execute these instructions are shown on the right. The mnemonic Rm, Rn or Ri indicates the address of a specified register. In this case, a second register file shown in FIG. **18** is defined and any of sixteen general-purpose registers, namely, four address registers **A0** through **A3**, four data registers **D0** through **D3** and eight extended registers **E0** through **E7**, may be specified. The second register file **120** further includes: a stack pointer (SP) **122**; a processor status word (PSW) **123** for holding internal status information and control information; and a program counter (PC) **124**.

FIG. **19** is a table of correspondence among name of a register specified during the execution of an instruction defined in the first instruction format, bit assignment on an instruction code specified in a register-addressing field, and number and name of a physical register to be accessed. In accordance with the first instruction format, each register-addressing field is composed of only 2 bits. However, in this case, there are sixteen general-purpose registers, each of which should be accessed using a 4-bit address. Accordingly, address conversion should be performed. For example, in accessing an address register **A0** and a data register **D1**, "1000" and "1101" should be produced as respective physical register numbers and then output to a file **121** of general-purpose registers.

FIG. **20** is a table of correspondence among name of a register specified during the execution of an instruction defined in the second instruction format, bit assignment on an instruction code specified in a register-addressing field, and number and name of a physical register to be accessed. In accordance with the second instruction format, each register-addressing field is composed of 4 bits, which is used as a physical register number as it is.

If variables are simply allocated preferentially to registers rather than memories as is done in a conventional compiler, then the data processor proposed by the present inventors in Japanese Laid-Open Publication No. 10-59680 poses the following problems:

1) A total length of instructions differs depending on whether variables, allocated to the first register file (including register resources), are processed in the first instruction format or variables, allocated to the second register file, are processed in the second instruction format. Accordingly, if these two types of variables are processed equally without prioritizing their allocation

at all, then the resulting code size of instructions cannot be minimized. That is to say, in a conventional compiler, it has not been taken into any consideration whether the variables should be preferentially allocated to the first or second register file. For example, if the variables are sequentially allocated to the second register file and processed in accordance with the second instruction format, then the resulting code size becomes longer. This is because the length of one instruction defined by the second instruction format is longer than that defined by the first instruction format.

2) In executing a set of instructions including a data transfer instruction from a memory to a register, the number of instructions where variables are processed in the first instruction format is larger than the number where the variables are processed in the second instruction format. But the total length of instructions in the first instruction format may be shorter than that in the second instruction format. Accordingly, even if variables are simply allocated preferentially to register resources rather than memories, the code size cannot be minimized.

SUMMARY OF THE INVENTION

An object of the present invention is providing a compiler that can produce an object program with a minimum code size for a processor of the type using different types of register resources and defining variable instruction lengths in accordance with the instruction formats.

In order to achieve this object, the compiler of the present invention produces an object program by allocating variables, which are referred to frequently, to register resources accessible in an instruction format with the shorter instruction length and by processing these variables in that instruction format.

Specifically, a compiler according to the present invention is adapted to translate a source program, including a plurality of instructions, into an object program. The compiler includes: first instruction length calculating means for calculating a total length of the instructions where variables for the source program are allocated to a first type of register resources in accordance with a first instruction format; and second instruction length calculating means for calculating a total length of the instructions where the variables are allocated to a second type of register resources in accordance with a second instruction format. The length of one instruction defined by the second instruction format is different from that defined by the first instruction format. The variables are allocated to respectively appropriate ones of the register resources based on the results of calculation derived by the first and second instruction length calculating means.

In one embodiment of the present invention, the variables are allocated to respectively appropriate ones of the register resources to make an ultimate total length of the instructions as short as possible based on the results of calculation derived by the first and second instruction length calculating means.

In another embodiment, some of the variables for the source program, which are referred to relatively frequently, are preferentially allocated to the first type of register resources accessible in the first instruction format.

In still another embodiment, in manipulating some of the variables for the source program, which have been allocated to the first type of register resources, the manipulation is described preferentially in the first instruction format.

In still another embodiment, some of the variables for the source program, which are referred to relatively frequently, and other variables used along with the former variables are preferentially allocated to the first type of register resources.

A system according to the present invention is adapted to minimize the code size of an object program executable on a computer. The object program has been translated from a source program using a compiler and the source program in-cludes a plurality of instructions. The compiler includes: first instruction length calculating means for calculating a total length of the instructions where variables for the source program are allocated to a first type of register resources in accordance with a first instruction format; and second instruction length calculating means for calculating a total length of the instructions where the variables are allocated to a second type of register resources in accordance with a second instruction format. The length of one instruction defined by the second instruction format is different from that defined by the first instruction format. The variables are allocated to respectively appropriate ones of the register resources based on the results of calculation derived by the first and second instruction length calculating means.

A computer-readable storage medium according to the present invention has stored thereon an object program that has been translated using a compiler from a source program including a plurality of instructions. The object program includes not only instructions described in a first instruction format using a first type of register resources, but also instructions described in a second instruction format using a second type of register resources. The length of one instruction defined by the second instruction format is different from that defined by the first instruction format. Each said instruction is identified as being in the first or second instruction format by a value in a particular field in the instruction.

According to the present invention, a processor, using different types of register resources and defining variable instruction lengths in accordance with the types of instruction formats, is supposed to be used. The inventive compiler calculates a total length of instructions in both cases where respective variables are allocated to a first type of register resources with the first instruction format and where the variables are allocated to a second type of register resources with the second instruction format. Based on these results of calculation, the variables are preferentially allocated to appropriate register resources to make the ultimate total length of instructions as short as possible. As a result, the compiler can produce an object program with a minimized code size.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. 1 is a block diagram illustrating a configuration of a compiler according to an exemplary embodiment of the present invention.

FIG. 2 is a block diagram illustrating a configuration of a resource allocator 3 in the compiler shown in FIG. 1.

FIG. 3 is a flowchart illustrating the flow of resource allocation processing performed by the resource allocator in the compiler shown in FIG. 1.

FIG. 4 illustrates an exemplary intermediate language program.

FIG. 5 illustrates life ranges and frequencies of reference of respective variables in the intermediate language program.

FIG. 6 illustrates an exemplary allocation of variables to respective resources and an associated machine instruction program in accordance with a conventional method.

FIG. 7 illustrates an exemplary allocation of variables to respective resources and an associated machine instruction program according to the present invention.

FIG. 8 illustrates another exemplary allocation of variables to respective resources and an associated machine instruction program according to the present invention.

FIG. 9 illustrates still another exemplary allocation of variables to respective resources and an associated machine instruction program according to the present invention.

FIG. 10 is a diagram illustrating a first instruction format (1) applicable to the compiler of the present invention run by a data processor.

FIG. 11 illustrates part of a list of specific instructions to be executed by the data processor in accordance with the first instruction format (1).

FIG. 12 is a diagram illustrating a first instruction format (2) executed by the data processor.

FIG. 13 illustrates part of a list of specific instructions to be executed by the data processor in accordance with the first instruction format (2).

FIG. 14 is a block diagram illustrating an arrangement of registers in a first register file in the data processor.

FIG. 15 is a table of correspondence illustrating respective relationships among names, numbers and types of registers in the register file and associated bit assignments where the data processor executes instructions in the first instruction format.

FIG. 16 is a diagram illustrating a second instruction format executed by the data processor.

FIG. 17 illustrates part of a list of specific instructions to be executed by the data processor in accordance with the second instruction format.

FIG. 18 is a block diagram illustrating an arrangement of registers in a register file in the data processor.

FIG. 19 is a table of correspondence illustrating respective relationships among names, numbers and types of registers in the register file and associated bit assignments where the data processor executes instructions in the first instruction format.

FIG. 20 is a table of correspondence illustrating respective relationships among names, numbers and types of registers in the register file and associated bit assignments where the data processor executes instructions in the second instruction format.

DESCRIPTION OF THE PREFERRED EMBODIMENTS

A compiler according to the present invention is adapted to produce a set of machine instructions using the data processor that is compatible with two different types of instruction formats as disclosed by the present inventors in Japanese Patent Application No. 10-59680 identified above. Specifically, the data processor can process a set of machine instructions by using different types of register resources and defining variable instruction lengths in accordance with the formats of the specific instructions.

Hereinafter, preferred embodiments of the present invention will be described with reference to FIGS. 1 through 5.

FIG. 1 illustrates a configuration of a compiler C included in a computer system S according to the present invention. The compiler C includes a syntax analyzer 1, an optimizer 2, a resource allocator 3 and a code generator 4.

The syntax analyzer 1 performs lexical analysis, syntax analysis and semantic analysis on a source program 5 stored as a file. The results of these analyses are output as an intermediate language program.

The optimizer 2 optimizes the intermediate language program in order to cut down on the size of an object program 6 ultimately generated and to shorten the execution time.

The resource allocator 3 identifies life ranges of respective variables in a program, allocates various resources, including registers and memories, to these variables within these life ranges, and also allocates an optimum instruction to an associated operation.

Based on the results of allocation obtained by the resource allocator 3, the code generator 4 converts the optimized intermediate language program into a set of machine instructions compatible with a target machine and outputs the instruction set as the object program 6.

It should be noted that the syntax analyzer 1, the optimizer 2 and the code generator 4 are all implemented as well-known software programs that run on a computer system, and the detailed description thereof will be omitted herein.

FIG. 2 illustrates a configuration of the resource allocator 3 of the compiler according to this embodiment of the present invention. As shown in FIG. 2, the resource allocator 3 includes: a life range determiner 31; first, second and third instruction length calculators 32, 33 and 34; a variable-to-resource allocator 35; and a controller 36. The life range determiner 31 determines the life range of each variable. The first instruction length calculator 32 calculates the total length of a set of instructions that have been used to allocate the variables to respective registers in accordance with the first instruction format, in which only part of the registers are available. The second instruction length calculator 33 calculates the total length of a set of instructions that have been used to allocate the variables to respective registers in accordance with the second instruction format, in which all of the register resources are accessible. The third instruction length calculator 34 calculates the total length of a set of instructions that have been used to allocate the variables to respective memories. Based on the respective results obtained by the life range determiner 31 and the first, second and third instruction length calculators 32, 33 and 34, the variable-to-resource allocator 35 ultimately decides to which resources the variables should be allocated. And the controller 36 controls the other sections of the resource allocator 3.

The resource allocation operation of the compiler having such a configuration will be described with reference to the accompanying drawings. FIG. 3 is a flowchart illustrating the flow of resource allocation processing under the control of the controller 36 in this embodiment.

An exemplary intermediate code program to be subjected to the resource allocation by the compiler is shown in FIG. 4. Intermediate codes s1 through s8 shown in FIG. 4 are collectively regarded as a basic block to be subjected to the resource allocation. For the sake of simplicity, the resource allocation within the basic block will be exemplified in the following description. However, a similar statement applies to resource allocation processing on a program of a size larger than that of the basic block.

Also, the register resource allocation is supposed to be performed under the following conditions for the illustrative purpose only.

1) Among the register resources accessible in the first instruction format (i.e., address registers A0 through A3 and data registers D0 through D3), the address registers A0 through A3 are supposed to be used as pointers, not to store data and variables thereon.

2) Among the register resources accessible in the first instruction format (i.e., address registers A0 through A3 and data registers D0 through D3), the data registers D0 through D3 are all supposed to be usable for storing and manipulating data and variables thereon. However, two out of the three data registers, namely, D0 and D1, are supposed to be used as work registers, not to allocate variables thereto.

3) Among the register resources accessible in the second instruction format (i.e., address registers A0 through A3, data registers D0 through D3 and extended registers E0 through E7), the extended registers E0 through E7 are supposed to be freely usable for data or address storage.

It should be noted that these conditions are defined herein to simplify the discussion, not to limit the scope of the present invention in any way.

Hereinafter, exemplary resource allocation processing on the intermediate code program shown in FIG. 4 will be described with reference to the flowchart shown in FIG. 3.

First, in Step S301, life ranges and frequencies of reference are examined for all the variables within the basic block. In this case, only the register resources accessible in the first instruction format, namely, A0 through A3 and D0 through D3, are available for the register allocation. Based on the results of this examination, the variable-to-resource allocator 35 checks whether or not each of the register resources, accessible in the first instruction format, is available for allocation. In this case, the key point is not how to allocate the variables to these registers, but that the register resource allocation is performed only on the register resources accessible in the first instruction format. This is because the length of an instruction defined by the first instruction format is shorter than that defined by the second instruction format in executing the same operation.

Next, in Step S302, it is determined based on the results of checking in Step S301 whether or not all the variables are allocable to only the register resources accessible in the first instruction format.

If the answer to the inquiry in Step S302 is "YES", then the variables are allocated in Step S303 only to the register resources accessible in the first instruction format to end the resource allocation processing.

In general, there are a large number of variables within a basic block. Accordingly, in most cases, some of the variables cannot be allocated to the register resources accessible in the first instruction format. If it has been determined in Step S302 that not all the variables can be allocated to the register resources accessible in the first instruction format, then the allocation of some variables to the register resources accessible in the first instruction format is prioritized. Hereinafter, this prioritized allocation processing will be described in detail.

FIG. 5 illustrates life ranges and frequencies of reference of respective variables within the basic block, which are used to determine which variables are allocable to the register resources accessible in the first instruction format. According to the results shown in FIG. 5, p1 and p2, which are variables referred to most frequently, are preferably allocated to the registers D2 and D3 among the registers accessible in the first instruction format. In this case, it is determined based on the number of accessible register resources and life ranges and frequencies of reference of respective variables which variables are allocated to these registers. Herein, since the registers D0 and D1 cannot be used under the conditions described above, the available register resources are D2 and D3, to which the two variables, referred to most frequently, are allocated.

Then, in Step S304, the variable-to-resource allocator 35 classifies the remaining variables, which have not been successfully allocated to the register resources accessible in the first instruction format, into the following two groups. Specifically, the first group consists of variables that should be allocated to other register resources accessible in only the second instruction format, i.e., extended registers E0 through E7, and the second group consists of variables to be allocated to memory resources. The combination of these variables, which are allocated to respective registers and memories, will be called a "first combination (resource allocation)" in the following description.

Furthermore, in Step S304, if it has been determined that some variables should be allocated to register resources in accordance with the first combination, then the controller 36 creates a second combination (instruction allocation). Specifically, the controller 36 determines whether the first instruction format should be adopted using the work registers D0 and D1 or the second instruction format should be adopted without using these two registers D0 and D1. Exemplary second combinations associated with particular first combinations using the extended registers E0 through E7 and the registers D2 and D3 are illustrated in FIGS. 7 through 9, which will be referred to in detail later. Since the total instruction lengths of all possible combinations are ultimately calculated, it does not matter which combination is selected at this point in time.

In Steps S305 through S307 to be described below, the total instruction lengths of the respective second combinations shown in FIGS. 7 through 9 are calculated.

Specifically, in Step S305, the total instruction length, where the remaining variables are allocated to other register resources in accordance with the first instruction format, is calculated. In Step S306, the total instruction length, where the remaining variables are allocated to other register resources in accordance with the second instruction format, is calculated.

Similarly, in Step S307, the total instruction length, where the remaining variables are allocated to memory resources, not the register resources, in accordance with the first instruction format, is calculated. In this case, an address usable for allocating a variable to an associated memory resource is assumed to be 16-bit absolute address, and a transfer instruction of the variable from a memory to a register is assumed to be composed of 3 bytes for the sake of simplicity. In addition, in Step S307, the total instruction length, where the remaining variables are allocated to memory resources, not the register resources, in accordance with the second instruction format, is also calculated. In this case, an address usable for allocating a variable to an associated memory resource is assumed to be 16-bit absolute address, and a transfer instruction of the variable from a memory to a register is assumed to be composed of 4 bytes for the sake of simplicity.

Next, in Step S308, the total length of the instructions for the entire basic block in accordance with this combination is calculated based on the results of Steps S305 through S307.

The same processing is performed on all the other possible combinations in Step S309. Finally, in Step S310, one of the combinations, resulting in the minimum length of the instructions for the basic block, is selected.

The compiler of the present invention minimizes the size of a machine-executable object program by performing these processing steps. This processing will be further detailed with reference to the accompanying drawings.

FIG. 6 illustrates the allocation of variables to respective resources in accordance with a conventional method and a

machine instruction program, in which respective instructions are assigned following the variable-to-register allocation. Specifically, in the example shown in FIG. 6, variables are sequentially allocated to some register resources accessible in the first instruction format and then to the other register resources without prioritizing the allocation of these resources. In this case, the total length of instructions for the basic block is 46 bytes.

FIG. 7 illustrates an exemplary allocation of variables to respective resources based on the life ranges and frequencies of reference of respective variables shown in FIG. 5 according to the present invention and a machine instruction program, in which respective instructions are assigned following the variable-to-register allocation. Specifically, in the example shown in FIG. 7, variables p1 and p2 are preferentially allocated to only the registers accessible in the first instruction format (i.e., registers D2 and D3). In this case, the total length of instructions for the basic block is 42 bytes, which is smaller than that shown in FIG. 6 by 4 bytes.

FIG. 8 illustrates another exemplary allocation of variables to respective register resources according to the present invention and a machine instruction program associated with the allocation. As in FIG. 7, the allocation of variables to only the registers accessible in the first instruction format is prioritized in FIG. 8. In the example shown in FIG. 8, however, the formats applied to some of the instructions are changed from second into first. As a result, the total instruction length itself for the basic block is 44 bytes, which is 2 byte larger than that shown in FIG. 7. However, as can be seen by comparing respective lengths of machine instructions between FIGS. 7 and 8 on an individual intermediate language basis, the lengths of machine instructions for the intermediate languages s1 and s8 are shorter in FIG. 8 than in FIG. 7 by one byte. This is because these intermediate languages s1 and s8 both include an instruction described in the first instruction format. In this example, the variables p1 and p2, which are referred to relatively frequently, are allocated to the registers D2 and D3 within the first register file **220** in the intermediate language s1. The variable t1, which is used with these variables p1 and p2, is also allocated temporarily to the work register D0 within the first register file **220**. Accordingly, the instructions mov D2,D0 and add D3,D0 are each described in the first instruction format with one byte.

FIG. 9 illustrates an ultimate allocation of variables to register resources and a machine instruction program in which the assignment of instructions has been optimized based on the results shown in FIGS. 7 and 8. In FIG. 9, each intermediate language is selected to have the shorter machine instruction length from the two types of intermediate languages shown in FIGS. 7 and 8. Specifically, the intermediate languages s2 through s7 shown in FIG. 7 are selected as the counterparts in FIG. 9, and the intermediate languages s1 and s8 shown in FIG. 8 are selected as the counterparts in FIG. 9. Although the intermediate languages s2 and s3 shown in FIG. 7 have the same machine instruction length as the counterparts shown in FIG. 8, those shown in FIG. 7 are selected. This is because processing can be performed faster in such a case since the number of instructions is smaller in FIG. 7 than in FIG. 8. As a result, the total code size is even smaller in FIG. 9 than in FIG. 7 by 2 bytes. In this manner, a machine instruction program with the smallest code size is produced for the intermediate language program shown in FIGS. 3 and 4.

In the machine instruction program shown in FIG. 9, the instructions mov D2,D0 and add D3,D0 of the intermediate language s1 and the instruction add 5,D0 of the intermediate language s8 are described in the first instruction format using the registers D0, D2 and D3 within the first register file **220**, while the other intermediate languages s2 through s7 are described in the second instruction format using the second register file **120**. And this machine instruction program is stored on a computer-readable storage medium.

In the foregoing embodiment, the respective variables p1, p2, and t1 through t8 for the basic block are allocated to appropriate ones of the registers D2, D3 and E0 through E7. If the number of variables to be allocated to respective resources exceeds the number of allocable register resources, then these variables are allocated to all of these registers and memory resources. In such a case, the total length of instructions, where variables are allocated to memory resources, is calculated in Step **S307** shown in FIG. 3. In allocating some of the variables to memory resources in this manner, the first instruction format is applied to a set of instructions including a data transfer instruction by temporarily using any of the registers included in the first register file **220**. As a result, although the number of instructions increases, the total length of instructions can be shortened and the ultimate code size may be reduced.

What is claimed is:

1. A compiler for translating a source program, including a plurality of instructions, into an object program, the compiler comprising:

first instruction length calculating means for calculating a total length of the instructions where variables for the source program are allocated to a first type of register resources in accordance with a first instruction format; and

second instruction length calculating means for calculating a total length of the instructions where the variables are allocated to a second type of register resources in accordance with a second instruction format, the length of one instruction defined by the second instruction format being different from that defined by the first instruction format,

wherein the variables are allocated to respectively appropriate ones of the register resources based on the results of calculation derived by the first and second instruction length calculating means.

2. The compiler of claim **1**, further comprising third instruction length calculating means for calculating a total length of the instructions where the variables are allocated to memories,

wherein the variables are allocated to respectively appropriate ones of the register resources based on the results of calculation derived by the first, second and third instruction length calculating means.

3. The compiler of claim **1**, wherein the variables are allocated to respectively appropriate ones of the register resources to make an ultimate total length of the instructions as short as possible based on the results of calculation derived by the first and second instruction length calculating means.

4. The compiler of claim **1**, wherein the first type of register resources are included in the second type of register resources.

5. The compiler of claim **4**, wherein the length of one instruction defined by the first instruction format is shorter than that defined by the second instruction format.

6. The compiler of claim **1**, wherein some of the variables for the source program, which are referred to relatively frequently, are preferentially allocated to the first type of register resources accessible in the first instruction format.

7. The compiler of claim **6**, wherein in manipulating some of the variables for the source program, which have been

allocated to the first type of register resources, the manipulation is described preferentially in the first instruction format.

8. The compiler of claim 6, wherein some of the variables for the source program, which are referred to relatively frequently, and other variables used along with the former variables are preferentially allocated to the first type of register resources.

9. A system for minimizing the code size of an object program executable on a computer, the object program having been translated from a source program using a compiler, the source program including a plurality of instructions, the compiler comprising:

first instruction length calculating means for calculating a total length of the instructions where variables for the source program are allocated to a first type of register resources in accordance with a first instruction format; and

second instruction length calculating means for calculating a total length of the instructions where the variables are allocated to a second type of register resources in accordance with a second instruction format, the length of one instruction defined by the second instruction format being different from that defined by the first instruction format,

wherein the variables are allocated to respectively appropriate ones of the register resources based on the results of calculation derived by the first and second instruction length calculating means.

10. A computer-readable storage medium having stored thereon an object program that has been translated using a compiler from a source program including a plurality of instructions,

wherein the object program includes not only instructions described in a first instruction format using a first type of register resources, but also instructions described in a second instruction format using a second type of register resources, the length of one instruction defined by the second instruction format being different from that defined by the first instruction format, and

wherein each said instruction is identified as being in the first or second instruction format by a value in a particular field in the instruction.

* * * * *